How to Write Word Game Programs for the Amstrad CPC 464, 664 and 6128

W. SIMISTER

HOW TO WRITE WORD GAME PROGRAMS FOR THE AMSTRAD CPC464, 664 AND 6128

OTHER BOOKS OF INTEREST

BP153 An Introduction to Programming the Amstrad CPC464 and 664
BP159 How to Write Amstrad CPC464 Games Programs
BP165 More Advanced Programming with the Amstrad CPC464, 664 and 6128
BP168 How to Write Practical Programs for the Amstrad CPC464, 664 and 6128
BP172 Useful Programming for the Amstrad CPC464, 664 and 6128

* * * * *

BP112 A Z80 Workshop Manual
BP152 An Introduction to Z80 Machine Code

HOW TO WRITE WORD GAME PROGRAMS FOR THE AMSTRAD CPC464, 664 AND 6128

by

W. SIMISTER

BERNARD BABANI (publishing) LTD
THE GRAMPIANS
SHEPHERDS BUSH ROAD
LONDON W6 7NF
ENGLAND

PLEASE NOTE

Although every care has been taken with the production of this book to ensure that any projects, designs, modifications and/or programs etc. contained herein, operate in a correct and safe manner and also that any components specified are normally available in Great Britain, the Publishers do not accept responsibility in any way for the failure, including fault in design, of any project, design, modification or program to work correctly or to cause damage to any other equipment that it may be connected to or used in conjunction with, or in respect of any other damage or injury that may be so caused, nor do the Publishers accept responsibility in any way for the failure to obtain specified components.

Notice is also given that if equipment that is still under warranty is modified in any way or used or connected with home-built equipment then that warranty may be void.

All the programs in this book were written and tested by the author using a model of the Amstrad CPC464 that was available at the time of writing in Great Britain and they are completely compatible with the Amstrad CPC664 and CPC6128. Details of the graphics modes may vary with versions of the machine for other countries.

First Published – December 1985

British Library Cataloguing in Publication Data
Simister, W
How to write word game programs for the Amstrad CPC464, 664 and 6128
1. Computer games
2. Amstrad Microcomputer Programming
I. Title
794.8'028'5425 GV1469.2

ISBN 0 85934 149 6

Printed and bounded in Great Britain by Cox & Wyman Ltd, Reading

FOREWORD

The excellent facilities of the Amstrad computers have been of great help in handling the words or letters in the many programs in this book. Included are ways to sort words alphabetically, to scramble them into anagrams, to select letters from words for various purposes, to locate missing letters from words, to place letters randomly on the screen, and even ways to help in the construction or solving of crossword puzzles.

The many routines used for all these different ways of handling words or letters are explained clearly, and are inserted into the programs in such a way that they can easily be extracted to be used in games of your own devising. One game, Canyon Crossing, shows how these different routines can be combined in a fairly long program.

One point should be stressed: if you want to be able to write programs for yourself it is essential to read, study, and thoroughly understand each chapter in turn. Even if you do not enter all the programs (but it is recommended that you do), each one should be understood before you go on to the next. All the programs have been written and used on the Amstrad CPC 464, but they are completely compatible with the Amstrad CPC 664 and CPC 6128.

Finally a routine has been included for those who have a DMP1 printer. It uses BASIC to produce on paper a copy of the screen in black and white. It is slow, but very sure.

W. Simister

PLEASE NOTE

The Publishers and Author would like to thank AMSTRAD Consumer Electronics plc for kindly supplying the photograph used in the cover illustration. Please note that the picture shown on the screen is not from a program contained in this book.

CONTENTS

Page

Chapter One
ANAGRAMMATISM 1

Chapter Two
MOVE A LETTER 10

Chapter Three
ALPHABETICAL WORD SORT 20

Chapter Four
LETTER EATER & HOW MANY WORDS 29

Chapter Five
GUESS THE WORD 40

Chapter Six
CROSSWORD SOLVER 48

Chapter Seven
CROSS-JOIN WORDS 55

Chapter Eight
THINKING OF A WORD SQUARE 64

Chapter Nine
CANYON CROSSING 75

Chapter Ten
PRINTING A SCREEN DISPLAY ON THE DMP1 . . . 92

Index 99

Chapter One

ANAGRAMMATISM

I cannot find the word Anagrammatism in my dictionary, but I take it to be the act of scrambling the letters of a word into a different order: forming an anagram. This very short program does just that. It is written so that ten different re-arrangements of an entered word are produced, but this can be altered from one to any number you wish. The program is at this chapter end, and is analysed very easily.

It consists of four short sections: Initialisation (title, inks used, Border and GOSUB); the Game (which produces the word to be scrambled); the 'shuffle-letters' section (in which the anagrams are produced and printed); and the printing of the screen layout. We will take them in the order in which they are used.

Initialisation is only a few lines: three are REM lines to display the title of the program. Line 40 defines the inks to be used. I prefer dark words on a light background, so have used the colours shown here. If you prefer light words on a dark background then alter those values shown: INK 0 is for the background, while INK 1 is for the lettering. You will see that defined in line 50, together with the BORDER. This is an area where personal likes are easily catered for. Spend a little time, when the program is entered and running, in altering those numbers (the second one after each INK), and you will find that it is easy to produce any combination of colours. The CLS at the end of line 50 ensures that the colours are as you wish on a cleared screen.

Line 60 GOSUB 3000 sends the computer to the screen layout section. Here, in line 3030, a FOR/NEXT loop produces the top and bottom line of a fancy border around the screen. 1 TO 25 STEP 24 ensures that only the top and bottom lines are printed. CHR$(238) from the character set was chosen, and was made into a string of 40 characters with: 'PRINT STRING$(40,CHR$(238));'. The semicolon (;) is essential, for without it the computer is triggered into scrolling up one line. Take out that semicolon, and watch the effect.

At this point it is worth emphasizing that one of the most usual faults with a beginner (and some more advanced programmers, too) is carelessness in entering the correct punctuation marks. If I get an error message when running a program I have written it is most frequently caused by an incorrect punctuation mark; usually a semicolon instead of a colon (:). My excuse (a poor one) is that these two are too close together on the Amstrad.

Line 3040 prints the left and right hand sides of the border, and is straightforward. 3050 is the Game title, and lines 3060 and 3070 print the border of the area where the scrambled words are to be printed. 3080 returns the computer to the line after GOSUB 3000: to the Game section.

Line 1030 contains a few words of instruction, and line 1040 asks for a word to be entered. This INPUT is located at 2,22 (near the bottom of the screen), and after the printed request is a semicolon (;) followed by the variable W$. This W$ will be the label of the word you enter. The next line is GOSUB 2000, and after the computer has gone to 2000 to race through the instructions there, it will RETURN (line 2180) to the line after GOSUB 2000.

The GOSUB/RETURN routine is one of the most useful facilities offered by a computer, for with its aid all the different routines can be kept separate, so that each can be referred to constantly. In the case of this program it need not have been used at this point, and the routine from 2030 to 2180 could have been inserted between lines 1040 and 1060, for this shuffle routine is used only once in this program. However, its separation helps you to see what each section does, and it is always wiser to use it when at all possible.

In line 2030 the variable J$ (which is the one used to label the forthcoming scrambled word) is DIMensioned to the LENgth of A$ (which later is made to equal W$). A$ is introduced into section 2000 so that if more than one word is being scrambled (in other programs) and given a different variable (G$ or P$, for instance) then line 2060 (A$ = W$) could be transferred to a line after the INPUT, and there A$ could be made equal to the other variable. This would enable the shuffle routine to remain self-contained, using A$ for its own lines.

Line 2050 defines the number of scrambled words to be produced. If lines 2050 and 2160 were removed then there would be only one scrambled word. If the 10 were altered to 100 there would be 100 of them. It is a simple repeat line between lines 2050 and 2160. Lines 2040 (Y=0) and 2070 (FOR X=1 TO LEN(A$)) are part of the LOCATE section, for they are used in line 2120.

At 2080 L=LEN(A$), and in 2100 R=INT(RND*L)+1. These are two variables essential to the shuffle formula in line 2110 J$=MID$(A$,R,1). These three lines together with line 2130, must be thoroughly understood in order to program words and letters.

First the L variable: L equals the LENgth of A$. A$ equals W$ (the word you entered), so L also equals the length of W$. Then, R equals the INTeger (the actual number) chosen randomly from the length of L (RND times L). Because the computer always starts numbering from 0, and therefore a seven letter word would be 0 to 6, we add +1 to the RND formula: R=INT(RND*L)+1.

Now we come to STRING handling, and the three key-words: MID$, LEFT$, and RIGHT$. The way each one functions is different. MID$ (followed by three items separated by commas within brackets); for example: MID$(A$,3,1) means: one letter at position three in the string A$. Put in another way: String A$, position 3, 1 letter.

Let us suppose that the word you entered was ANTIDOTE: then MID$(A$,3,1) means T; MID$(A$,5,2) would mean DO; MID$(A$,3,4) would mean TIDO. Make sure you understand this properly. Use a different word than ANTIDOTE, and experiment.

LEFT$ uses a different formula: LEFT$(A$,3). There are only two expressions within the brackets. For instance: LEFT$(A$,3) in reference to ANTIDOTE means ANT. LEFT$ selects the letters to the left of (and including) 3. LEFT$(A$,1) means A. LEFT$(A$,7) means ANTIDOT.

RIGHT$ is similar to, but opposite to, LEFT$, for, using ANTIDOTE: RIGHT$(A$,1) would mean E; RIGHT$(A$,7) would mean NTIDOTE; RIGHT$(A$,4) would mean DOTE. RIGHT$ selects the number given from the right (including

that number).

Study these three keywords very thoroughly. MID$ will pick out any letter, or number of letters, from anywhere in the string. LEFT$ starts from the left, and picks out the letters up to and including the one numbered. RIGHT$ starts from the right, and counts back to and including the one numbered. Don't forget that *including.*

If you have really grasped that, then the explanation of the important lines of this program should now be completely understood. Line 2110 J$=MID$(A$,R,1) means that the position of the single letter selected is governed by R, which is the one randomly chosen in line 2100.

2120 LOCATE X+13,Y+8: PRINT J$ refers back to line 2040, where Y=0, and line 2070, where X=1 TO the length of A$. So X=13 means (on the first pass of the FOR/NEXT loop X), 1+position 13 (that is 14); Y+8 means 8. Therefore LOCATE 14,8:. On the second pass it would be 15,8:, and on the third pass, 16,8:. Thus the letters selected are placed one after the other to the LENgth of A$ on row 8.

Now comes the tricky bit. Line 2130 A$=LEFT$(A$,R–1) +RIGHT$(A$,L–R). This line is there to ensure that on its second pass through A$ to select a letter at random from A$, it does it from an A$ that no longer contains the previous letter chosen. Try reading that again.

Let us suppose that the word is ANTIDOTE, and that line 2110 selected the I with its R variable. Therefore, before the sequence is gone through again, we wish A$ to equal ANTDOTE (with the I missing). Line 2130 does that extraction of the I with the aid of LEFT$ and RIGHT$. On that first pass (through loop X) I was randomly chosen by R; I is the fourth letter, so R had become equal to 4. Therefore LEFT$(A$,4–1) would mean ANT.

The RIGHT$ part of the formula would read RIGHT$ (A$,L–4): that is the LENGTH at the right of 4, but minus the single letter numbered 4 from the right; and so it would mean DOTE. The plus sign in the middle of the formula means that A$ now equals ANT+DOTE, so on the second pass through loop X the I would not be included.

If you have not previously handled strings in this way, it will be best if you study this matter thoroughly. In the last

few paragraphs is all the information needed to become thoroughly familiar with handling strings in this manner on the Amstrad.

When line 2140 NEXT X has used up the length of A$, the computer goes to 2150 Y=Y+1, and at 2160 NEXT Q it is sent back to line 2050 FOR Q=1 TO 10, and goes right through the whole procedure again for the next scrambled word; but this time placing it on a line lower down the screen.

If you have followed the explanation given up to now it should be clear that the elimination of lines 2050 and 2160 would prevent more than one anagram being printed, as mentioned before. Incidentally, now that you understand all the complex sequences required to print one anagram, and then watch the speed with which the Amstrad produces ten of them, you will have a much greater respect for the speed of Locomotive BASIC as used by the Amstrad.

When all ten anagrams are printed the computer goes on to line 2170, which is a line to erase the request for a word at the screen bottom (notice how SPC(30) is used to do this), and then returns to the line after GOSUB 2000; to line 1060.

There you are asked (in a flashing red colour) if you want to have another go. This flashing red colour was arranged in line 40, where INK 3 has been made to alternate between red and the background colour by adding the background colour after a comma; INK 3,6,23. Erase the second comma and the 23 from the end of line 40, and the question will then be plain red.

If the answer to the flashing question is Y (yes) then at 1110 the screen is cleared, and the program is RUN again. There are other ways of doing this should this question be asked in a different program. Then line 1110 could be: FOR Y=8 TO 17:X=13:LOCATE X,Y:PRINT SPC(22): NEXT Y:LOCATE 2,22:PRINT SPC(30):GOTO 1030. This would be useful in a game where scores were being kept from game to game, but the PRINT SPC(?) statements would, of course, have to be located at different places. If N (no) is answered then the screen is cleared, instructions given to print "Thank you", and the game ends.

This short program has a number of purposes: first is the pattern in which the program is laid out; with separate sections,

each headed with REM statements to explain what is held in each section, and to demonstrate the GOSUB/RETURN routine.

Secondly, to explain the principles involved in using 'Strings' of letters or words, and in manipulating them. This is one of the most important Basic functions, and a knowledge of it is essential for most word games and manipulations.

Thirdly, to provide a small section that can be extracted for use in other programs. Following the program at this chapter end are 9 extra lines: 5000 to 5080. They have been taken from the main program, and are lines 1040 (with a slight alteration), and lines 2070 to 2140.

If, when these nine lines have been entered, you RUN 5000, a question will appear on the screen: "Enter a word". When this is entered a single anagram of your word will be printed. The program is complete in those nine lines, and with suitable alterations could be used anywhere. If those extra lines are saved to tape as an extra to the main program, they will not affect its functioning at all.

```
10 REM ************************
20 REM **** ANAGRAMMATISM ****
30 REM ************************
40 INK 0,23:INK 1,1:INK 2,11:INK 3,6
,23
50 BORDER 16:PAPER 0:PEN 1:CLS
60 DIM J$(LEN(A$))
70 GOSUB 3000
1000 REM ******************
1010 REM ***** GAME *****
1020 REM ******************
1030 PEN 2:LOCATE 3,7:PRINT "ENTER":
LOCATE 3,9:PRINT "a WORD":LOCATE 3,1
1:PRINT "up to 22":LOCATE 3,13:PRINT
 "letters":LOCATE 3,15:PRINT "lon9."
:PEN 1
1040 LOCATE 2,22:INPUT "ENTER A WORD
";W$
1050 GOSUB 2000
```

```
1060 LOCATE 2,22:PEN 3:INPUT "ANOTHE
R GO(Y/N)";Q$:PEN 1
1070 IF Q$="Y" OR Q$="y" THEN 1110
1080 IF Q$="N" OR Q$="n" THEN CLS
1090 LOCATE 13,10:PRINT "THANK YOU"
1100 GOTO 1100
1110 CLS:RUN
1120 LOCATE 2,20:END
2000 REM ************************
2010 REM *** SHUFFLE LETTERS ***
2020 REM ************************
2030 DIM J$(LEN(A$))
2040 Y=0
2050 FOR Q=1 TO 10
2060 A$=W$
2070 FOR X=1 TO LEN(A$)
2080 L=LEN(A$)
2090 J$=""
2100 R=INT(RND*L)+1
2110 J$=MID$(A$,R,1)
2120 LOCATE X+13,Y+8:PRINT J$
2130 A$=LEFT$(A$,R-1)+RIGHT$(A$,L-R)
2140 NEXT X
2150 Y=Y+1
2160 NEXT Q
2170 LOCATE 2,22:PRINT SPC(30)
2180 RETURN
3000 REM ******************
3010 REM ***** SCENE *****
3020 REM ******************
3030 PEN 2:FOR Y=1 TO 25 STEP 24:X=1
:LOCATE X,Y:PRINT STRING$(40,CHR$(23
8));:NEXT Y
3040 FOR X=1 TO 40 STEP 39:FOR Y=2 T
O 24:LOCATE X,Y:PRINT CHR$(238):NEXT
:NEXT:PEN 1
```

```
3050 PEN 2:LOCATE 4,3:PRINT "*** A N
 A G R A M M A T I S M ***":PEN 1
3060 PEN 2:FOR Y=6 TO 19 STEP 13:LOC
ATE 12,Y:PRINT STRING$(26,CHR$(238))
:NEXT Y
3070 FOR X=12 TO 37 STEP 25:FOR Y=7
TO 18:LOCATE X,Y:PRINT CHR$(238):NEX
T Y:NEXT X:PEN 1
3080 RETURN
5000 CLS:INPUT "ENTER A WORD";A$
5010 FOR X=1 TO LEN(A$)
5020 L=LEN(A$)
5030 J$=""
5040 R=INT(RND*L)+1
5050 J$=MID$(A$,R,1)
5060 PRINT J$;
5070 A$=LEFT$(A$,R-1)+RIGHT$(A$,L-R)
5080 NEXT X
```


*** A N A G R A M M A T I S M ***
ENTER
a WORD
up to 22
letters
long.
EEEERTSLHVNS
EELSVRSEEHNT
LSETHRESNEEV
HEERELTEVSNS
HTLESEVENSRE
HEEVERESLSTN
LSRIEVNHEEES
RINEEEVSHSEL
SELNREHETVES
HESELVSERENT
ENTER A WORD? NEVERTHELESS

Chapter Two

MOVE A LETTER

In this chapter is a program based on the shuffling routine used in Chapter One. It is the simple game of letters arranged in a square, completely out of order, which you have to move one by one into alphabetical order. There is one blank space, but instead of the usual four by four arrangement, this is a game played with five by five, employing 24 letters and a space. When you have entered it, and you try your hand at re-arranging the letters, you will find that it takes a great deal of time and patience. Also, there is an unkind line below the square which registers how many moves you have made. Don't be surprised if your first try runs you well into double figures; it takes many hundreds to finish the game – if you ever do. The main aim of the program is to show you how a game is constructed.

The layout of the screen display is fairly straightforward, but if you are designing one for a program of your own it is essential to use a Plotting Board on which to draw it first. The one I use is of Indian ink on a sheet of white Formica, but a stout sheet of cardboard would do. Make the sheet about 12 x 8¼ inches, and on that draw a pattern of squares: 41 lines horizontally, and 26 lines vertically. Start the lines one inch from the edge, and separate the lines by a quarter inch. I work from the top edge and the left-hand edge when measuring and marking. The pattern of squares thus drawn will occupy 10 x 6¼ inches.

When this is drawn (in fairly black ink) start entering the X,Y numbers: X (for text of all sorts) along the top, and Y (for text) down the left-hand side (1 to 25 downwards). Make sure that the numbers (1,2,3 etc.) are opposite the space between the lines – not on the line ends. This is important, for letters and numbers fall into the squares – not on the lines.

Having completed the X,Y text numbers (used for LOCATE), the next task is the pixel numbers used by ORIGIN and MOVE. These run along the bottom edge (X,

left to right), and up the right-hand side (Y, bottom to top). These numbers must be at the line ends (not spaces), and rise in steps of 16: 0, 16, 32, 48, 64 and so on to 640 along the bottom, and the same numbers from bottom to top (to 400) up the right-hand side. When using the board you will have to estimate any numbers between those written for more detailed drawing. If this is your main interest then I recommend a board twice as big each way, with double the number of lines numbered in steps of 8 for graphics. Of course the text numbers at the top and left-hand side would then have to be carefully positioned on every other line end. Think that over.

When this board is finished, and the ink dry, obtain some thin sheets of paper (through which the lines will show) to lay over the board, and on which your rough layouts can be pencilled in. With an outfit like this designing a program layout is easy. For this present program you only have to copy down the listing at this chapter end, but I will describe the reasoning behind the layout as we progress.

There are six sections in this program: the first being the initialising of INK colours and a GOSUB 3000. In section 3000 is the screen layout, placed at the program end for convenience. It can be altered to suit yourself. After the title: MOVE A LETTER, come some instructions (down the right-hand side) followed by a small block of letters to illustrate what you will be aiming at in the game.

Next comes the line locating where the moves are to be shown, and following this is the actual board (or box) in which the game is played. I used my plotting board to plan all this first, so it was easy then to place the commands and locations as I wrote the program. However, the design of the board did take some time, and I shall explain the thinking behind it.

A close block of 24 letters and a space could have been used quite easily, but I felt it was better to space out the letters a little so that a framework of lines could be used to contain them. Putting a letter at every other space left an empty text space between each letter without any risk of erasure as letters were moved. Similarly, a border was drawn around the block to enhance its appearance. The whole thing

then occupied a block of fourteen text spaces each way.

Lines 3070 to 3160 were used to draw the border, the lines enclosing the 25 squares, and the location numbers at the top and sides. To draw the lines enclosing the 25 squares a tidy formula has been used, contained in lines 3110 to 3140. Six lines were required each way, each separated by 32 pixels. The method used here makes Q equal to 1 TO 6, and then in the ORIGIN statement uses 32 times Q (32 times 1; 32 times 2; etc.), plus 184 (the pixel at which to start. This is equivalent to using a FOR/NEXT loop of: FOR X=216 TO 376 STEP 32, but is more convenient, for lines are drawn both ways with one loop. That little routine (lines 3110 to 3140) is worth copying out, and using whenever you have to draw crossed lines of equal spacing and number.

The rest of the lines in section 3000 are straightforward. In line 3180 a dark line is drawn between border and screen all around, for it makes the whole layout tidier. It is only ornamental. 3190 RETURN sends the computer back to the lines after GOSUB 3000, to 500.

Here the shuffle formula from the previous game is used, with some slight alterations. Line 530 makes A$ equal to 24 letters of the alphabet plus a space, for in this game A$ is fixed. The space has been placed among the letters, but could have been at the end or beginning. It is not so likely to be omitted during the entering of the listing in its present place. So far as the computer is concerned the space is just one more character to be entered when it is selected by R.

540 DIM J$(5,5) starts the sequence of locating the chosen letters in the spaces provided. 550 and 560 establish the number in each row and column. 570 to 600 were explained in the previous chapter, and 610 is the LOCATE and PRINT procedure. 2 times X (which is 1 TO 5) means 2 TO 10 STEP 2, and plus 13 means starting at column 13; so in effect that X part of LOCATE means 15 TO 23 STEP 2. Go over that again until it is perfectly clear.

If it is not clear then remember that you are more intelligent than a computer, but you may not be thinking logically. The computer does *not* think, but it reacts to signals that are sent to it in the most logical way possible. It reads the messages sent to it quite literally; we are inclined to allow

extraneous thoughts to interfere with our logic.

The Y part of the LOCATE section is dealt with similarly: it means 8 TO 16 STEP 2, using that same reasoning: 2 times Y+6. PRINT J$(Y,X) ensures that the computer uses that reasoning when printing.

Line 620 is exactly as it was in the previous game, and gets rid of the selected letter (from A$) on each pass of the loop. That brings up another point: the positioning of the two Y and X loops. They are placed with the Y loop outside the X loop so that the letters are printed left to right in the top row, left to right in the 2nd row, and onward. If the X was outside the Y they would be printed down the first column, down the second, and so on. This is because the inner of the two loops is always run through completely before going to the second pass of the outer loop.

The computer then moves on directly to the game section at 1000, where it is necessary for four INPUT entries to be made. You must enter the letter you wish to move; its position (before the move) (top number and side number); and whether it is to move left (L), right (R), up (U), or down (D). This involves a decision about which way to program the entries: the four entries can be made separately (with a single key-press each); or they can be made as a four-item INPUT followed by ENTER. In the first example it is easy to make a mistake, or forget what the fourth entry should be. In the second example it is possible to check, and to alter if necessary, any item before pressing ENTER, so this second method was chosen.

This composite method of INPUT entry is very convenient, and not used as often as it might be. Notice the manner in which the INPUT statement in the program is entered, line 1030. The location; then the Print statement started with INPUT instead of PRINT; the statement itself; a semi-colon (;); and four variables separated by commas. The statement asks for LET. (the letter to be moved); top no. side no. (present location); and then a letter to indicate which way you want it to move. Take care to use the forms of punctuation that are used in the listing – which is taken from a working program.

Because the first request is for a letter, its variable, Z$,

must be a string. The second and third requests are for numbers, so they can have single variables (X & Y). The fourth item is a letter again, so this must be a string (K$ is chosen in this instance). These must be separated by commas, and the ENTER key is only pressed when all four are entered, and have been checked. This makes it easier to play, for there is only one Enter (although of four items) for each move.

Immediately the ENTER key is pressed the computer adjusts its variables memory to store those four values, and they remain in memory until altered. Line 1040 GOSUB 2000 sends the computer to the conversions, where it learns that X, and then Y have taken on new values. It is RETURNed in line 2130, to 1050, where, at the new X,Y numbers it prints a space (to erase the letter to be moved).

At once it goes to the next line, 1060 GOSUB 2500, and at 2530 onward finds that K$ has now altered the value of X (or Y). 2570 sends it back to 1070, where it locates the new X,Y position, and there prints Z$ (which was the first of the letters ENTERed). The move has been made. Line 1080 erases the original INPUT question (with SPC(35)); line 1090 adds one to the 'moves' score, and prints it in its proper place; and line 1100 directs the computer back to 1030 to the INPUT once again.

As mentioned at the beginning of this chapter, this game takes so long to finish that it requires endless patience. If you wish to make it shorter you can alter it to 4 x 4 letters, instead of 5 x 5; it merely requires some alterations to the shuffle letter routine. At line 530 erase all letters after 0 (remove P to X). In line 540 make DIM J$(5,5) into (4,4). At lines 550 and 560 alter lines 1 TO 5 into 1 TO 4. The game will then play in the rows and columns 1 to 4, leaving row 5 and column 5 blank.

To make the game completely 4 x 4 would entail some major alteration to the screen section in many lines, but if you wish to do that you must work out the changes yourself. With the knowledge you have now gained that should be easy, but do retain a copy of the 5 x 5 game on tape first. The lines requiring any change would be 3050 and 3070 to 3160.

Sub-routines that can be extracted from this program to use in your own programs are: the one mentioned earlier

(lines 3110 to 3140 for printing a square of lines); lines 530 to 640 (to produce a shuffle of letters and print them at separated places in a square pattern); and lines 1030 to 1070 (for an INPUT including more than one item).

Other matters to note carefully are the way in which letters or words must have a string variable, while numbers have a letter only (not a string); the use of double inverted commas (lines 2530 to 2560) for the subject of strings, and of no inverted commas (lines 2030 to 2120) for non-string variables.

Note also the use in lines 2530 to 2560 of upper and lower case letters in the IF statements, for this avoids errors. Note that in line 1070 the print statement is PRINT UPPER$(Z$) in order to cover the inadvertent use of lower case letters when entering. All these things-to-note are an essential part of programming.

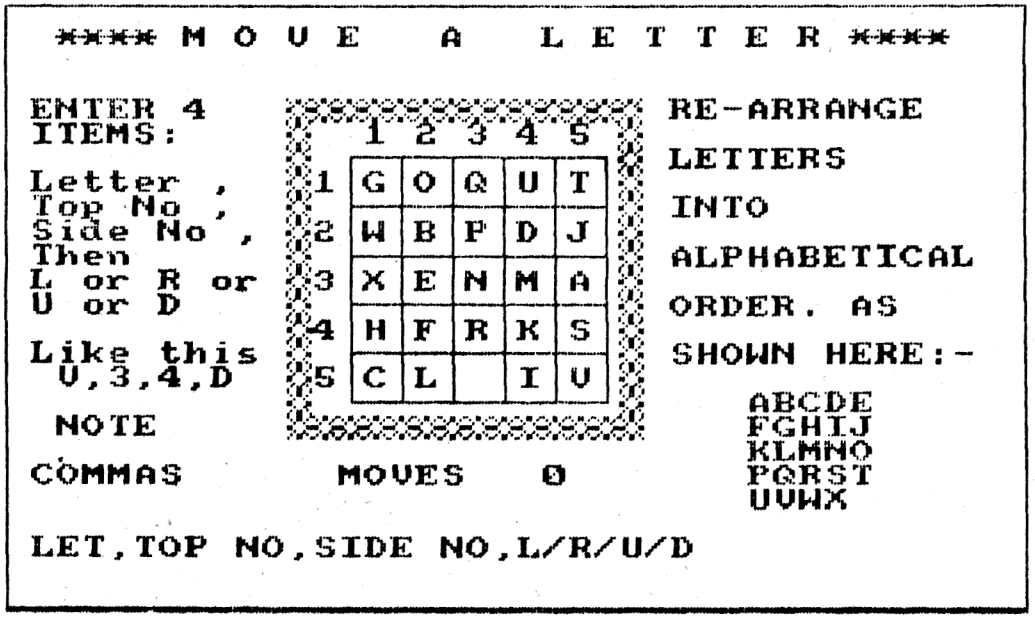
This latter is a potent cause of errors. I will give an example. On many occasions when entering a taped program that had previously worked perfectly I found it would not work. After much searching I then found I was using the computer in lower case mode, instead of CAPS MODE, and that the lower case letters would not trigger the actions I wanted. It must be remembered that the computer makes a proper distinction between lower and upper case letters; they have totally different code numbers. Therefore this must be guarded against, always.

```
10 REM ************************
20 REM **** MOVE A LETTER ****
30 REM ************************
40 INK 0,25:INK 1,9:INK 2,18:INK 3,6
50 BORDER 18:PAPER 0:PEN 1:CLS
60 GOSUB 3000
500 REM ************************
510 REM *** SHUFFLE LETTERS ***
520 REM ************************
530 A$="ABCDEFG HIJKLMNOPQRSTUVWX"
540 DIM J$(5,5)
```

```
550 FOR Y=1 TO 5
560 FOR X=1 TO 5
570 L=LEN(A$)
580 J$=""
590 R=INT(RND*L)+1
600 J$(Y,X)=MID$(A$,R,1)
610 LOCATE 2*X+13,2*Y+6:PRINT J$(Y,X
)
620 A$=LEFT$(A$,R-1)+RIGHT$(A$,L-R)
630 NEXT X
640 NEXT Y
1000 REM **********************
1010 REM **** GAME MOVES ****
1020 REM **********************
1030 LOCATE 2,23:INPUT "LET,TOP NO,S
IDE NO,L/R/U/D";Z$,X,Y,K$
1040 GOSUB 2000
1050 LOCATE X,Y:PRINT " "
1060 GOSUB 2500
1070 LOCATE X,Y:PRINT UPPER$(Z$)
1080 LOCATE 2,23:PRINT SPC(35)
1090 moves=moves+1:LOCATE 21,20:PRIN
T moves
1100 GOTO 1030
2000 REM **********************
2010 REM *** CONVERSIONS ***
2020 REM **********************
2030 IF X=1 THEN X=15
2040 IF X=2 THEN X=17
2050 IF X=3 THEN X=19
2060 IF X=4 THEN X=21
2070 IF X=5 THEN X=23
2080 IF Y=1 THEN Y=8
2090 IF Y=2 THEN Y=10
2100 IF Y=3 THEN Y=12
2110 IF Y=4 THEN Y=14
```

```
2120 IF Y=5 THEN Y=16
2130 RETURN
2500 REM ********************
2510 REM *** MOVE LETTER ***
2520 REM ********************
2530 IF K$="L" OR K$="l" THEN X=X-2
2540 IF K$="R" OR K$="r" THEN X=X+2
2550 IF K$="U" OR K$="u" THEN Y=Y-2
2560 IF K$="D" OR K$="d" THEN Y=Y+2
2570 RETURN
3000 REM ********************
3010 REM ***** SCREEN *****
3020 REM ********************
3030 LOCATE 3,2:PEN 3:PRINT "**** M
O V E   A   L E T T E R ****":PEN 1
3040 LOCATE 27,5:PRINT "RE-ARRANGE":
LOCATE 27,7:PRINT "LETTERS":LOCATE 2
7,9:PRINT "INTO":LOCATE 27,11:PRINT
"ALPHABETICAL":LOCATE 27,13:PRINT "O
RDER. AS":LOCATE 27,15:PRINT "SHOWN
HERE:-"
3050 LOCATE 30,17:PRINT "ABCDE":LOCA
TE 30,18:PRINT "FGHIJ":LOCATE 30,19:
PRINT "KLMNO":LOCATE 30,20:PRINT "PQ
RST":LOCATE 30,21:PRINT "UVWX"
3060 LOCATE 14,20:PEN 3:PRINT "MOVES
":PEN 1:moves=0:LOCATE 21,20:PRINT m
oves
3070 LOCATE 12,5:PEN 2:PRINT STRING$
(14,CHR$(238)):LOCATE 12,18:PRINT ST
RING$(14,CHR$(238))
3080 FOR P=0 TO 11
3090 LOCATE 12,6+P:PRINT CHR$(238):L
OCATE 25,6+P:PRINT CHR$(238)
3100 NEXT P:PEN 1
3110 FOR Q=1 TO 6
```

```
3120 ORIGIN 32*Q+184,136:DRAW 0,160
3130 ORIGIN 216,104+32*Q:DRAW 160,0
3140 NEXT Q
3150 LOCATE 15,6:PRINT "1 2 3 4 5"
3160 LOCATE 13,8:PRINT "1":LOCATE 13
,10:PRINT "2":LOCATE 13,12:PRINT "3"
:LOCATE 13,14:PRINT "4":LOCATE 13,16
:PRINT "5"
3170 LOCATE 2,5:PRINT "ENTER 4":PRIN
T " ITEMS:":PRINT:PRINT " Letter ,":
PRINT " Top No ,":PRINT " Side No ,"
:PRINT " Then":PRINT " L or R or":PR
INT" U or D":PRINT:PRINT " Like this
":PRINT "  V,3,4,D"
3175 PRINT:PRINT "  NOTE":PRINT:PRIN
T " COMMAS"
3180 ORIGIN 0,0:DRAW 639,0:DRAWR 0,3
99:DRAWR -639,0:DRAWR 0,-399
3190 RETURN
```


**** M O U E A L E T T E R ****
ENTER 4 ITEMS:
Letter ,
Top No ,
Side No ,
Then
L or R or
U or D
Like this
U,3,4,D
NOTE
COMMAS
1 2 3 4 5
1 G O Q U T
2 W B P D J
3 X E N M A
4 H F R K S
5 C L I U
MOVES 0
RE-ARRANGE
LETTERS
INTO
ALPHABETICAL
ORDER. AS
SHOWN HERE:-
ABCDE
FGHIJ
KLMNO
PQRST
UVWX
LET,TOP NO,SIDE NO,L/R/U/D

Chapter Three

ALPHABETICAL WORD SORT

This short program has been designed to show a number of different functions; among them the use of WINDOW to separate off a portion of the printed text, so that a list of words can be entered, and will SCROLL without affecting other words on that screen. It is a program that sorts words into alphabetical order, and will sort numerals also, provided they are of even length: 364, 027, 002, 244, etc., or 29, 36, 08, 13, 02, etc.

The words are entered one after the other in response to the INPUT, but the program can be altered easily to a program in which the words are put into a DATA statement to be READ. This is the main purpose of this chapter, to demonstrate the changing of a program, but first we will deal with the original.

After the initialisation of INKs and BORDER comes the title; a couple of words of instruction, and a thick border around the screen. Then, at line 100, is 'WINDOW#1,2,39,7, 24' which defines the area of a window excluding the title and border: X equals 2 TO 39 and Y equals 7 TO 24.

If, after the program is entered and taped, you type LIST#1, and ENTER, then the listing will appear and scroll inside that border without affecting it in any way. Similarly, a part of the program will be printed only in that area. This ensures that if you decide to enter more than 15 words (the limit defined at present) they will scroll in that area. But more of that later.

Lines 500 to 630 are used to enter the words you wish to arrange in alphabetical order. A is made to equal −1, so that the first word entered will be number 1. Line 540 defines the list of words to be 15: 'DIM B$(15)'; but this number can be altered to suit your own convenience – provided you also alter line 620 (IF A=15) in the same way. A$="*" is there to establish A$ as a string variable. Then a WHILE/WEND loop starts. This is similar to a FOR/NEXT loop in that it makes whatever is between WHILE and WEND repeat, but

different in that it is not for a fixed number of times. In this case it repeats until 'nothing' is entered: WHILE A$ (is not equal to) " ".

The INPUT (into WINDOW 1) is given the variable A$ (line 580). In line 590 is an IF statement to ensure that if the number of words entered is less than 15 then a second ENTER will move the computer into the SORT routine. 600 A=A+1 is a line to make A increase in number as each word is entered. 610 B$(A)=A$ means that B$ (the list of words) (A) (the number of words) is equal to A$ (the INPUT).

This section has been described in detail because it will be one of those to be altered when we wish to make a different program: one that will accept words from DATA, rather than from INPUT.

The SORTing section (1000–1150) is a standard one, and takes B$ (from the previous section) to represent the list of words. Because the list is complete, and its number known (A), the next line is the beginning of a FOR/NEXT loop: FOR B=0 TO A–1. This is the outside one of three FOR/NEXT loops in this section (B, D, and E): D and E being separated (not nested) but both inside B. It is in the three loops combined that the sorting occurs, and it should be studied thoroughly until completely understood. It entails some mind-boggling thinking while following it through, but it is worth it. At the end of that section B$(B) holds the list of words alphabetically in its memory.

In section four (1500–1600) is the printing section. This also will be altered for the second program, but only for its X location, because in section 2 the INPUTs are printed down the left-hand side of the screen; in section 4 they are started at X=20 (to the right). Also, they are printed in a different colour.

Lines 1530 and 1540 establish the X,Y numbers, while the change of INK colour is in 1550. Note the way it is done: 1550 PEN#1,2. The HASH sign (#) must be used on each occasion that PEN, LOCATE, and PRINT are used for it to operate correctly, and must be followed by a comma as shown in the listing.

A FOR/NEXT loop is then started: FOR B=0 TO A in line 1560. The first word is printed (blue) in line 1570. Notice

LOCATE#1 and PRINT#1. Then there is B$(B) (which is 0 TO A as in section 3) plus CHR$(7). CHR$(7) is a BEEP, so that when added as here a BEEP is sounded each time a word is entered. This is a most useful command for games, and while not necessary here, it does add something to the actual production of the list.

In line 1580 X is equal to 20; Y is made equal to Y+1; and a pause is arranged with 1 to 200. In 1590 is NEXT B, so that the next word is printed. When the list has been completed (B=0 TO A is finished) the computer goes on to PEN#1,1 (so that the future printing in the window is PEN 1), and in 1600 goes to 1600. This last line obviates any message on the screen, and ESC has to be used to Break.

This small program is excellent for occasions when it is convenient to enter the words to be shuffled one at a time, but when a longer list is needed, and it is possible to forget what has been entered, a READ/DATA method of entry is better. We will alter this program to do just that, but make sure first that it is safely put on tape just as it stands.

Then, on the computer, lines 90, 530 to 630, and 1030 to 1150 can be deleted; 90 because the instructions in that line are no longer relevant; 530 to 630 because we shall replace them with entries to READ DATA from lines at the end; and 1030 to 1150 because we shall use a slightly different sorting technique this time. It is important to know that there are more ways than one to sort words.

When these lines have been deleted then use RENUM to start each section with new numbers as follows: what was line 10 must now be 3000, so type in directly: RENUM 3000,10,10 and ENTER. In case you have not yet realised all the renumbering facilities on the Amstrad CPC464, 664 or 6128, here are some details: Of the three numbers after RENUM, the first is the new line number; the second is the old number from which you want it to start; and the third is the spacing (I normally use 10 for this). All RENUMbering is from the second number to the end of the program. To RENUMber the different sections in this new program will entail four RENUM orders: RENUM 3000,10,10 ENTER; LIST to see it is correct; the next section you want to start at 3500, but it is now at 3090, so: RENUM 3500,3090,10

ENTER; LIST; the section you want at 4000 is now at 3530, so: RENUM 4000,3530,10 ENTER; LIST; the section you want at 4500 is now at 4030, so: RENUM 4500,4030,10 ENTER; and LIST. The new DATA section will be at 5000.

The complete listing for this new program is at this chapter end, but RENUM will have saved you many entries. Line 3080 is now altered a little: the WINDOW can now be one line higher, thus giving more room for words. The new routine for entering words starts at 3530 with DIM K$(55). This number (55) can be altered, but must always be more than the number of words that are stored in the DATA lines.

Line 3540 READ N: this is a number variable, and applies to the number of words to be sorted. It must be exactly correct, and words are picked up (READ) from line 5030 DATA 25 (at present). Line 3550 FOR I=1 TO N takes its number from N, and the FOR/NEXT loop operates just as many times as the number in line 5030. Make sure you really understand this, for many errors can crop up if it is ignored.

The next line: 3560 READ K$(I) is a string variable, and can READ words from the DATA at line 5040 onward – up to the number of N (in 5030). Line 3570 NEXT I keeps its repeating until the number is reached, and then goes to the next section.

It will be noticed that I is used as a variable here. All the other variables have been changed also, because this program is to be held in memory as well as the first program (we will MERGE them later), and it is best not to confuse the memory with similar variables in two programs at the same time.

Those five lines (3530–3570) will extract the words for shuffling, but they must have some DATA from which to do this. Section 5000 contains just three lines, but any number could be added, which is why the DATA is put at the end. Whatever number of words are included must be accurately put into line 5030 in place of the present 25. I have used simple Christian names in the DATA for this demonstration, but they could be anything. The comma is the functional divider between words (items or blocks of words) in every case, but there is no comma between DATA and the first

word. If you wanted to use a list of combined Christian and Surnames, like Robinson, Bill; or Jones, Richard, you must not put a comma between the Surname and Christian name, as is the usual custom, because the computer would then separate them as distinct pieces of DATA. The usual way is to leave a space, as I have done for one demonstration pair of words. You might also use a full stop (.).

With the section at 5000 entered you can return to 4000, for the SORTing. The first line prints the word SORTING outside the WINDOW: notice that LOCATE and PRINT do not have the hash sign following them. This time the actual SORT is done with two FOR/NEXT loops, an I inside an S loop. The lines 4070–4100 do the actual work, and are relatively easy to follow. Make sure you understand how this section works before you proceed.

The fourth section at 4500 deals with the printing, but this time X is equal to 2, bringing the list to the left of the screen. Y equals 1, so the words will be started at the top of the WINDOW, using PEN#1,2 (line 4550) for colour. LOCATE and PRINT are on the same line, and are each followed by the hash sign, and at the end is the BEEP. Line 4580 sets Y to equal Y+1, so that the next word is printed one line lower. FOR T=1 TO 200 causes a slight pause, and 4590 sends the program back to line 4560 until all of I has been used in the FOR/NEXT loop. Then the WINDOW PEN is returned to 1 again. Line 4600 prevents a Break sign marring the screen.

Programmers who have a printer, and who wish to use this second program for a list or index make, can add a REM to line 4570, and then add another line: 4575 PRINT#8,K$(I). When RUN there will be the usual pause while the computer does the SORTing, and then the list is printed on the printer.

It is easy to make an address or/and telephone list: ROBINSON DAVID.3 MEMORY ST.NANTWICH(467352) could be such an entry. Remember that spaces and full stops (and brackets) can be used within an entry, but not a comma. If you use more letters than the line can hold on the screen, the computer will use the next line also, but on the printer more characters can be used without going onto another line. I use the DMP1 printer, and find that the

addition of WIDTH 80 before running the index program does the trick.

RUN 3000, and ENTER will produce the second program, while RUN and enter produce the first. In these two programs are many routines to use elsewhere: WINDOW in lines 100 and 3080 (and examples of using them later in each program); an INPUT routine using WHILE/WEND (530–630); a READ/DATA routine (lines 3530–3570 coupled with lines 5030 onward); two distinct sorting methods (1030–1150 and 4030–4140); and a routine for printing a list of words from memory (with a BEEP) (lines 1530–1600 and 4530–4600). These are all routines that can be extracted, and (when adapted) used for other programs.

```
10 REM ************************
20 REM ***** WORD  SORT *****
30 REM ************************
40 INK 0,23:INK 1,1:INK 2,2:INK 3,10
50 BORDER 5:PAPER 0:PEN 1:CLS
60 PEN 2:LOCATE 4,2:PRINT "***** ALP
HABETICAL WORD SORT *****":PEN 1
70 ORIGIN 0,0:DRAW 639,0:DRAWR 0,399
:DRAWR -639,0:DRAWR 0,-399
80 ORIGIN 2,2:DRAW 635,0:DRAWR 0,395
:DRAWR -635,0:DRAWR 0,-395
90 LOCATE 3,4:PRINT "Enter words.
    Use CAPITALS ONLY":LOCATE 9,5:PR
INT "Use ENTER twice to end."
100 WINDOW #1,2,39,7,24
500 REM ************************
510 REM ***** ENTER WORDS *****
520 REM ************************
530 A=-1
540 DIM B$(15)
560 A$="*"
570 WHILE A$<>""
580 INPUT #1,A$
```

```
590 IF A$="" THEN 630
600 A=A+1
610 B$(A)=A$
620 IF A=15 THEN PRINT #1,"ARRAY NOW
 FULL"
630 WEND
1000 REM ********************
1010 REM ***** SORTING *****
1020 REM ********************
1030 LOCATE #1,20,1:PRINT #1,"SORTIN
G..."
1040 FOR B=0 TO A-1
1050 C$=B$(B)
1060 C=B
1070 FOR D=B+1 TO A
1080 IF C$>B$(D) THEN C$=B$(D):C=D
1090 NEXT D
1100 IF C=B THEN 1150
1110 FOR E=C-1 TO B STEP-1
1120 B$(E+1)=B$(E)
1130 NEXT E
1140 B$(B)=C$
1150 NEXT B
1500 REM ****************************
1510 REM ***** PRINTING WORDS *****
1520 REM ****************************
1530 X=20
1540 Y=3
1550 PEN #1,2
1560 FOR B=0 TO A
1570 LOCATE #1,X,Y:PRINT #1,B$(B)+CH
R$(7)
1580 X=20:Y=Y+1:FOR T=1 TO 200:NEXT
T
1590 NEXT B:PEN #1,1
1600 GOTO 1600
```

```
3000 REM ************************
3010 REM ***** WORD  SORT *****
3020 REM ************************
3030 INK 0,23:INK 1,1:INK 2,2:INK 3,
10
3040 BORDER 5:PAPER 0:PEN 1:CLS
3050 PEN 2:LOCATE 4,2:PRINT "***** A
LPHABETICAL WORD SORT *****":PEN 1
3060 ORIGIN 0,0:DRAW 639,0:DRAWR 0,3
99:DRAWR -639,0:DRAWR 0,-399
3070 ORIGIN 2,2:DRAW 635,0:DRAWR 0,3
95:DRAWR -635,0:DRAWR 0,-395
3080 WINDOW #1,2,39,6,24
3500 REM ************************
3510 REM ***** ENTER WORDS *****
3520 REM ************************
3530 DIM K$(55)
3540 READ N
3550 FOR I=1 TO N
3560 READ K$(I)
3570 NEXT I
4000 REM ********************
4010 REM ***** SORTING *****
4020 REM ********************
4030 LOCATE 2,4:PRINT "SORTING..."
4040 FOR S=1 TO N-1
4050 M=0
4060 FOR I=1 TO N-S
4070 IF K$(I)<=K$(I+1) THEN 4120
4080 X$=K$(I)
4090 K$(I)=K$(I+1)
4100 K$(I+1)=X$
4110 M=1
4120 NEXT I
4130 IF M=0 THEN 4530
4140 NEXT S
```

```
4500 REM ***************************
4510 REM ***** PRINTING WORDS *****
4520 REM ***************************
4530 X=2
4540 Y=1
4550 PEN #1,2
4560 FOR I=1 TO N
4570 LOCATE #1,X,Y:PRINT #1,K$(I);+C
HR$(7)
4580 X=2:Y=Y+1:FOR T=1 TO 200:NEXT T
4590 NEXT I:PEN #1,1
4600 GOTO 4600
5000 REM ****************
5010 REM ***** DATA *****
5020 REM ****************
5030 DATA 25
5040 DATA JOHN,HARRY,HERBERT,JOSEPH,
JAMES,CHARLES,JACK,HENRY,ADAM,BILL,R
OBERT,FRED,IVOR,JOSHUA,LARRY
5050 DATA SAMUEL,THOMAS,GEORGE,PAUL,
BENJAMIN,RALPH,NICHOLAS,JENKINSON PE
TER,DANIEL,DAVID
```

Chapter Four

LETTER EATER & HOW MANY WORDS

The program LETTER EATER places the letters of the alphabet in randomly chosen places, and then provides a ball which is moved around the screen to erase the letters one by one in alphabetical order. It is very simple, but the writing of it involved many problems that can crop up in other programs, so a description of those problems, and how they were tackled, should help the reader when devising games of his own.

It has been written with sections, but each section follows consecutively, so after the initialisation of INKs and the BORDER the screen is drawn. Line 130 is the title, and 140–150 draw a display box to contain the game; 160 puts a dark line between border and screen, and 170 gives some instructions. Line 180 makes a small box to contain the score.

The third section (200 to 370) places the letters in position. These are at random positions, and in random colours. Placing them should have been a simple problem, but difficulties arose, and the first solution was found to be unsuitable. Just four lines were used for this first solution:

```
230 FOR A=65 TO 90
240 C=INT(RND*3)+1
250 X=INT(RND*34)+1:Y=INT(RND*15)+1
260 PEN C:LOCATE X+3,Y+3:PRINT CHR$(A):
    NEXT A:PEN 1
```

In line 230 (65 to 90) are the CHR$ numbers of A to Z. Line 240 makes C equal to 1, 2, or 3, and is later used for the PEN number. In line 250 X is made equal to a randomly chosen number between 1 and 34 (the number of X spaces available), while Y is equal to 15 randomly chosen numbers.

Line 260: after PEN C (3 colours), LOCATE X+3,Y+3 locates the position on the screen, and then comes PRINT CHR$(A) followed by NEXT A. CHR$(A) is first CHR$(65),

and on subsequent passes is CHR$(66), (67), (68), and so on to (90).

These four lines appeared to do the job at first, but occasionally it was noticed that some letters were missing. The explanation was this: On going back through line 250 a previously used X,Y number was being chosen, and one letter was printing on top of another, thereby erasing the previous letter. Sometimes two or three letters were so erased, and that wouldn't do at all.

In that 4-line version the X and Y numbers were random, and the letters were consecutive from A to Z. It was decided that X should be a continuous progress across the screen, and the letters and Y numbers would be randomized. This was done so that the letters could be made into a string, for then the normal way of extracting a previously used letter (to prevent its second use) could be used. This is in the present line 350. There are now 15 lines in this section, but it works perfectly. It is worth studying, for it is typical of programming problems that can crop up, so we will examine it line by line.

Line 230 makes A$ equal to A to Z. In 240 'X=3' allows the area used for letters to start at 3. In line 250 a FOR/NEXT loop starts, and continues until the LENgth of A$ (26 letters) is used up. Line 260 gives C the random value of 1, 2, or 3 (for the PEN colours). Line 270 T=INT(RND*1.5)+1 needs a little more explanation. There are 36 possible X positions in the area used, but only 26 letters, so if line 330 were X=X+1 then all the letters would be on the left-hand side of the area. I wished to ensure that the move from one X position to the next X position was just a little more than one space (that is: occasionally one space, and occasionally 2 spaces). Lines 270 and 330 do just that.

Line 280 makes L the LENgth of A$; line 290 makes R the variable number chosen to select one of the letters; line 300 makes J$ equal to the letter thus randomly chosen, and line 320 prints it – at the X number chosen earlier, and the Y position chosen randomly in line 310 (where there are 17 available) in either of 3 colours (PEN C). There is a line (340) to see that X cannot go too far to the right, and then we come to 350, which is mentioned earlier.

350 A$=LEFT$(A$,R–1)+RIGHT$(A$,L–R) has been explained at length in Chapter One, so if you missed that, go back and read it now. Used in a similar position (and with relevant string variables) it is a standard way of extracting a character from a string. 360 is NEXT P, and 370 sends the PEN colour back to 1. That second section: PLACE LETTERS, is complete on its own, and is worth keeping handy. There is a way of doing this easily if you have a Printer, otherwise you should write it down in a notebook (few of us have minds that can carry all such details in memory). Using a printer I first enter directly: PRINT#8, "PLACE RANDOM LETTERS, RANDOM COLOURS, RANDOM PLACES" and ENTER. Then use: LIST 230–370 #8 and ENTER. This puts the item on paper for you, and a collection of such scraps can be invaluable.

The next section (at 400) is used to move the little ball that erases the letters. The area within which it can move is: X, 3 to 38, and Y, 3 to 20 (that is inside the border). Lines 430, and 490 to 520, confine its movements to that area. Notice how it is done with 'less-than' or 'more-than' signs with IF. Line 530 CALL &BD19 is there to make things move smoothly.

CHR$(231) is the ball; CHR$(32) is a space. The moves are made with lines 450–480. The right-pointing arrow is number 1, the left is 8, down is 2, and up is 0. We will take just one move: Press right arrow, and line 450 sends the computer to 560, where, at X,Y, a space is printed. Then X is made to equal X+1, and (ignoring the Z moves for the moment) the computer returns to the line after 450, and traverses down to 540, where, at the new X,Y number, the ball is re-printed.

Notice the pause (FOR Q=1 TO 50:NEXT Q) in line 540. This ensures that the printed ball is in view longer than the space. Lines 450 to 480 all work the same, with the computer swopping from one to the other at a furious pace.

Now the Z. This is there merely to count the number of moves it takes to erase the whole alphabet. Z is first set to 0 (line 430) to establish it as a variable. Then at each move Z is increased by 1, and printed in the small rectangle. This is a simple way of scoring, and sufficient for this game.

If you want to save this movement routine on the Printer, first remove Z=0 from line 430, and the Z sections from lines 560 to 590 (Z=Z+1:LOCATE 32,23:PRINT Z). Remember to leave the RETURN at the end of each of the 4 lines. The routine could then be titled 'MOVEMENT IN ALL DIRECTIONS'.

That was a small game, and one not likely to hold your interest for long. The next game: HOW MANY WORDS, is very different. The program is short (but can be lengthened), and is a mind-boggling game, because from less than 30 letters (carefully arranged) as many as 27 words can be made, using only sequences of consecutive letters. You will understand this better when you have entered and run this program (it is at the chapter end).

After the introductory lines detailing the colours, comes GOSUB 3000 where the scene is drawn. In this the title, some instructions, and four rectangles are drawn, and then the computer returns to GOSUB 2000, where a simple random choice is made of any of three items. Note the way this is done in lines 2030 to 2060: just four lines. Using RANDOMIZE TIME ensures a different starting point for each random choice, but because in this program only three choices are given (at present) there are occasional repeats of the same choice. R is made to equal either 1, 2, or 3. Then three lines tell the computer which line to go to in either case. I will explain the increase to a greater number than 3 later on.

Assuming that R=1, then the computer goes to 2200, where A$ is made equal to a list of words. The making up of this list of words had better be explained here, for it involves quite a lot of thinking, and some help from a dictionary.

A simple word is chosen: CANAL in this instance; and then, using one or more of that word's last letters, a second word is chosen. In this case it was ALLOW, so we now have CANALLOW. LOWEST was the next word chosen, giving CANALLOWEST. Using just that much it is now possible to make from it eleven words: CAN, AN, CANAL, ANAL, ALL, ALLOW, LOW, OWE, WE, WEST, LOWEST: eleven words from eleven letters is not bad.

Now A$ actually includes (in this case) 28 letters, and I have found it contains 27 words. This is the basis of the

television game 'NOW YOU SEE IT', although that game includes innumerable variations on that theme. Perhaps later, when you have absorbed the information in this book, you can make up a new game of your own, based on one or more of those variations.

To return to this program. A$ (in the case of R=1) equals 28 letters. Now to allow for a form of scoring, we wish to know how many words A$ contains. So, after line 2210, which prints A$ in the rectangle provided, we form another variable (AAA$) at line 2220. This AAA$ contains all the words I have been able to make from A$, separated by the backward-slope mark (above CTRL). This is chosen instead of a comma because in this context it is accepted by the computer as just a character.

Line 2230 is GOSUB 2800, and at that line is a sequence to count the number of words in AAA$. This is done by counting the backward-slopes, and adding 1 during a FOR/NEXT loop of M (the LENgth of AAA$). 2840 is the RETURN (to 2230), which itself is a RETURN to 1030. The computer will now have in memory AAA$, A$, and A., as well as other variables.

At 1030 (the Game section) with 1040 is an INPUT where you type in the words you can make from A$. This new INPUT is AA$. This facility of the Amstrad (of being able to differentiate between A, A$, AA$, and AAA$) is most useful in this case. The aim of the INPUT is to enter words (divided by a backslope so that they can be counted) and not use ENTER until you have made sure you have typed in every word you can see.

When this is ENTERed line 1050 prints (in the small blue rectangle) the number of words you should have entered (A). 1060 GOSUB 1190 takes the length of your entry (at line 1190) as L=LEN(AA$), and in the next three lines gives it the variable P, and returns it to 1070. There P is printed in the first of the red rectangles, and in 1080 the score is printed in the form P IN A (your entry and what you should have entered). Lines 1120 to 1160 offer you another go, by first erasing your entry, providing another INPUT (Y/N), and acting on that.

It is a surprising fact that even though you have entered

the program, including the lines of words it is possible to find, when it comes to playing the game you may forget some – even though there are only three sets of words. Just imagine how difficult it would be if there were more sets. And that is your next task: to add more lists. There are four lines to each set: 2200–2230, 2240–2270, and 2280–2310, and there is plenty of room to add more (between 2310 and 2800). However, you must also alter 2030 in the R=INT section, changing the 3 into whichever number you alter it to, and adding after line 2060 more lines of: 'IF R= '. Line spaces have been left for these additions.

I have previously described how to form a list of words, so now start making some more of these sets; eventually the program could contain anything up to a dozen or more lines of words. If you like mind games you will get more fun out of enlarging this program than you will out of playing it.

In 'HOW MANY WORDS' there are fewer separate routines to separate out for future use, but there are many uses of GOSUB (to separate out small routines) to serve as examples. Also note lines 2030–2060. This routine, to select a line randomly, can be extremely useful in many games, so is worth noting. Also, the two similar routines (lines 1190–1220 and 2800–2840) are well worth keeping on file.

```
10 REM ************************
20 REM *** EAT THE LETTERS ***
30 REM ************************
40 INK 0,26:INK 1,2:INK 2,18:INK 3,6
50 BORDER 15:PAPER 0:PEN 1:CLS
100 REM ********************
110 REM ***** SCREEN *****
120 REM ********************
130 LOCATE 2,2:PRINT STRING$(12,CHR$
(238));" LETTER-EATER ";STRING$(12,C
HR$(238))
```

```
140 FOR Y=3 TO 20:FOR X=2 TO 39 STEP
 37:LOCATE X,Y:PRINT CHR$(238):NEXT
X:NEXT Y
150 LOCATE 2,21:PRINT STRING$(38,CHR
$(238))
160 ORIGIN 0,0:DRAW 639,0,1:DRAWR 0,
399:DRAWR -639,0:DRAW 0,-399
170 LOCATE 2,23:PRINT "Move BALL wit
h arrows.":LOCATE 2,24:PRINT "Erase
letters alPhabetically."
180 ORIGIN 488,24:DRAW 128,0,3:DRAWR
 0,32:DRAWR -128,0:DRAWR 0,-32
200 REM **********************
210 REM *** PLACE LETTERS ***
220 REM **********************
230 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
240 X=3
250 FOR P=1 TO 26
260 C=INT(RND*3)+1
270 T=INT(RND*1.5)+1
280 L=LEN(A$)
290 R=INT(RND*L)+1
300 J$=MID$(A$,R,1)
310 Y=INT(RND*17)+1
320 PEN C:LOCATE X,Y+3:PRINT J$
330 X=X+T
340 IF X>38 THEN X=38
350 A$=LEFT$(A$,R-1)+RIGHT$(A$,L-R)
360 NEXT P
370 PEN 1
400 REM **********************
410 REM ***** MOVEMENT *****
420 REM **********************
430 X=3:Y=3:Z=0
440 WHILE 1
450 IF NOT INKEY(1) THEN GOSUB 560
```

```
460 IF NOT INKEY(8) THEN GOSUB 570
470 IF NOT INKEY(2) THEN GOSUB 580
480 IF NOT INKEY(0) THEN GOSUB 590
490 IF X<3 THEN X=3
500 IF X>38 THEN X=38
510 IF Y<3 THEN Y=3
520 IF Y>20 THEN Y=20
530 CALL &BD19
540 LOCATE X,Y:PRINT CHR$(231);:FOR
Q=1 TO 50:NEXT Q
550 WEND
560 LOCATE X,Y:PRINT CHR$(32):X=X+1:
Z=Z+1:LOCATE 32,23:PRINT Z:RETURN
570 LOCATE X,Y:PRINT CHR$(32):X=X-1:
Z=Z+1:LOCATE 32,23:PRINT Z:RETURN
580 LOCATE X,Y:PRINT CHR$(32):Y=Y+1:
Z=Z+1:LOCATE 32,23:PRINT Z:RETURN
590 LOCATE X,Y:PRINT CHR$(32):Y=Y-1:
Z=Z+1:LOCATE 32,23:PRINT Z:RETURN
```

```
10 REM ****************************
20 REM ***** HOW MANY WORDS *****
30 REM ****************************
40 INK 0,25:INK 1,9:INK 2,2:INK 3,6
50 BORDER 25:PAPER 0:PEN 1:CLS
60 GOSUB 3000
70 GOSUB 2000
1000 REM ***********************
1010 REM ***** MAIN GAME *****
1020 REM ***********************
1030 LOCATE 3,18:PRINT "    ENTER WO
RDS DIVIDED BY \ "
1040 PRINT:INPUT AA$
```

```
1050 LOCATE 36,12:PRINT A
1060 GOSUB 1190
1070 LOCATE 17,15:PRINT P
1080 LOCATE 31,15:PRINT P;"IN";A;
1120 LOCATE 3,18:PRINT SPC(36):LOCAT
E 1,20:PRINT SPC(39):PRINT SPC(39):P
RINT SPC(39):PRINT SPC(39)
1130 LOCATE 3,18:INPUT "ANOTHER GO(Y
/N)";Z$
1140 IF Z$="" THEN 1140
1150 IF Z$="Y" THEN CLS:RUN
1160 IF Z$="N" THEN CLS
1180 STOP
1190 L=LEN(AA$)
1200 P=1:FOR M=1 TO L
1210 IF MID$(AA$,M,1)="\" THEN P=P+1
1220 NEXT M:RETURN
2000 REM ****************************
2010 REM ***** LETTER STRINGS *****
2020 REM ****************************
2030 RANDOMIZE TIME:R=INT(RND*3)+1
2040 IF R=1 THEN 2200
2050 IF R=2 THEN 2240
2060 IF R=3 THEN 2280
2200 A$="CANALLOWESTIMEATTACKNOWLEDG
E"
2210 LOCATE 4,12:PRINT A$
2220 AAA$="CANAL\CAN\AN\ANAL\ALL\ALL
OW\LOWEST\LOW\OWE\WE\WEST\TIME\ME\ME
AT\EAT\AT\ATTACK\TACK\ACKNOWLEDGE\KN
OWLEDGE\KNOW\NO\NOW\OWL\LEDGE\LED\ED
GE"
2230 GOSUB 2800:RETURN
2240 A$="BELOVEDUCATEGORYEASTERNICOT
INE"
2250 LOCATE 4,12:PRINT A$
```

```
2260 AAA$="BE\LOVE\LOVED\BELOVED\EDU
CATE\DUCAT\ATE\CATEGORY\AT\CAT\EGO\G
O\OR\RYE\YEA\YEAST\EAST\EASTERN\AS\A
STERN\STERN\TERN\NICOTINE\COT\TIN"
2270 GOSUB 2800:RETURN
2280 A$="CANNOTWITHSTANDINGLEANCESTR
AL"
2290 LOCATE 4,12:PRINT A$
2300 AAA$="CAN\AN\ANNO\NO\NOT\CANNOT
\NOTWITHSTANDING\WITHSTAND\WITH\STAN
D\STANDING\AN\AND\DIN\IN\DINGLE\INGL
E\GLEAN\LEAN\AN\ANCESTRAL"
2310 GOSUB 2800:RETURN
2800 L=LEN(AAA$)
2810 A=1:FOR M=1 TO L
2820 IF MID$(AAA$,M,1)="\" THEN A=A+
1
2830 NEXT M
2840 RETURN
3000 REM ****************************
3010 REM ***** SCREEN DISPLAY *****
3020 REM ****************************
3030 PEN 3:LOCATE 3,2:PRINT "* * * *
 * HOW MANY WORDS * * * * *":PEN 1
3040 PEN 2:PRINT:PRINT "  EXAMPLE:
C A N N O T  contains:        CAN,AN
,ANN,ANNO,NO,NOT,CANNOT":PEN 1
3050 PRINT "   How many words can yo
u make using     any consecutive se9
uence of letters     in the followin
g list of letters?"
3060 PEN 2:PRINT "   Don't use ENTER
 until all words are   entered":PEN
1
3070 ORIGIN 40,200:DRAW 496,0,2:DRAW
R 0,32:DRAWR -496,0:DRAWR 0,-32
```

```
3080 ORIGIN 552,200:DRAW 80,0,2:DRAW
R 0,32:DRAWR -80,0:DRAWR 0,-32
3090 LOCATE 2,15:PRINT "words entere
d":ORIGIN 232,152:DRAW 96,0,3:DRAWR
0,32:DRAWR -96,0:DRAWR 0,-32
3100 LOCATE 24,15:PRINT "SCORE:":ORI
GIN 472,152:DRAW 160,0,3:DRAWR 0,32:
DRAWR -160,0:DRAWR 0,-32
3110 RETURN
```

Chapter Five

GUESS THE WORD

A method of forming an anagram was given in Chapter One. In this chapter is a simple game, GUESS THE WORD, using that anagram routine. In it you can enter any number of guesses at the word, and if you're baffled, and ask for help, a letter, or letters, will be printed. While the program is simple the game is not, for each word among the 50 in DATA has nine or more letters, and in its anagram form gives no clue to its true nature. The full listing is at this chapter end.

As well as being a demonstration of how to use anagrams in a game, this program illustrates well the principle of using WINDOW. With its help any number of guesses can be entered, and the list of words entered will scroll up without affecting the screen layout of other sections. When you reach that part of the program explanation make every effort to understand how it is used, for that facility of the Amstrad 464, 664 and 6128 is most useful.

The format, or structure, of this program is straightforward, with just four sections, each separated by REM titles. Lines 40 and 50 define the colours to be used in a formula that has become a habit of mine. The four INKs to be used (MODE 1) are always defined in line 40, and in 50 is the BORDER colour, with PAPER and PEN followed by CLS, which in this position ensures that if any colour is changed in these two rows, then the use of RUN will produce that colour.

Then come two lines (60 and 70) that define the two WINDOWs used. The first is for the scoring (a small one on the right), and the second (on the left) reaches from near the title to the screen bottom for the entry of guesses. Notice how these WINDOWs are entered: with a comma after hash 1 (or 2) (hash is the name given to #), and then the four numbers (separated by commas) that define the WINDOW areas. The first one is X, 22 to 40, and Y, 6 to 12; just seven lines that are 19 character spaces long. The second is X, 1 to

21, and Y, 4 to 25; 22 lines each 21 spaces long. Later we shall see how hash 1 and hash 2 are used in practice. You may have noticed that in hash 1, PEN 2 is used; take note of how this is entered, with the comma after hash 1.

Lines 80 and 90 print the title and an arrow pointing to a box. In that box will appear the anagram. CHR$(243) is the arrow. Then come lines 100 to 130, each printing in hash 1 (that is WINDOW 1). Line 100 is an example of how they should be entered: LOCATE(space)hash 1(comma)5(comma) 1(colon)PRINT(space) hash 1(comma)(inverted-commas) SCORE(inverted-commas). Notice that it is essential to use hash and a comma in both LOCATE and PRINT commands. Note also that the LOCATE numbers have no reference to the ordinary X,Y numbers. Each WINDOW has its own numbers: for hash 1 they are X, 1 to 19, Y, 1 to 7. This SCORE is printed at 5,1 (in terms of that WINDOW's numbers). This is a point often missed, so study it until you have grasped it; it makes WINDOWS easy. Lines 140–150 print some instructions.

The section at 200 to 480 deals with the moves in the game, but it starts with GOSUB 600, section 3, so we will go to that one first. This has lines 600 to 790, but it uses section 4 as well, lines 800 to 870.

Lines 630 to 790 use a simple anagram-making routine, but unlike the anagram in Chapter One, it takes (READS) its word from a list of DATA, instead of from an INPUT. It is a routine well worth keeping for future use, so make sure you understand just how it works.

Line 630 defines the DIMensions of J$ as the LENgth of A$. Lines 640 and 650 are used to give the variable B a random value. A FOR/NEXT loop starts in 660, using C=0 TO B. Line 670 selects from the DATA list (lines 800 onward) the word selected by B, and 680 finishes that loop. READ W$ (line 670) means that the word selected at random is given the variable W$.

Line 690 sets the value of Y to 0, and in 700 A$ is made equal to W$ (the chosen word); so at this stage both A$ and W$ equal that word. A FOR/NEXT loop starts at 710: X equals the LENgth of A$. 720 is L=LEN(A$). At first sight it might seem that these two lines should be reversed (710

becomes 720, and 720 becomes 710); but this is not so. 710 is a FOR line, and NEXT (at 780) returns to that FOR line. That would be ignoring line 770, which makes A$ one character less on each pass, and so needs re-defining; therefore the lines are in their correct order now. Make sure you understand this point.

At line 730 J$ is set to equal nothing; 740 makes N the variable carrying a random value of L (present LENgth of A$). The +1 is used at the end of random lines because the computer always starts counting from 0. In 750 J$ is now made equal to a single letter, selected from A$ by: MID$(A$,N,1). The last number inside the brackets ensures that only one letter is chosen; N decrees which letter, and A$ decides where it is taken from.

760 LOCATE X+25 needs explaining. The 25 is where the position starts, but X is different on each pass, increasing by one character each time, for X is the control in the FOR/NEXT loop. Y had been set to 0, but is now +2, so that moves 2 lines down. PRINT J$ is fairly clear: it prints the letter.

At 770 comes that standard extraction line that was described in Chapter One. It pulls that printed J$ out of A$, and sends A$ on with one letter less. 780 sends the computer back to 710 (the FOR line), and the process is repeated until X is exhausted. Then 790 sends the computer back to 240, the line after GOSUB 600.

Before leaving the third and fourth sections of the program it will be best to deal with DATA. First of all line 650 B=RND*50, refers to the number of pieces of DATA. If you alter the number of pieces in DATA you must make line 650 correspond, otherwise you will get error messages.

DATA is always entered in one way: LINE NUMBER(space) DATA(space)WORD(comma)WORD(comma)WORD (with no comma at the end).

There are five lines of DATA at present in this program, and each line contains ten words (merely for easy counting; they could be any number). They were obtained from a dictionary, and are ten words from A, ten words from B, and so on up to E. If you want to alter this program (use shorter words, or more words) it can be done simply by altering those DATA lines (and line 650, of course). For my own use I have

produced two versions of this game, one with four- or five-letter words (suitable for youngsters), and one with longer words for those who like to stretch their minds. In both cases I have increased the DATA to over 200 words, which makes them into really good games. The five lines of DATA in this present program contain a lot of tough words, so that each anagram that comes up will tax most people.

To return to the actual program at line 240. This is another example of printing in a WINDOW. Line 240 could be added to the beginning of 250, for it is there only to make the computer jump one line. Notice that 'hash 2' has to be used with every command (LOCATE, PRINT, and INPUT). Line 260 gives your entry (your guess at the word) the variable G$, followed by G$=UPPER$(G$), in case you are not typing in CAPS MODE. Then come six lines of IF conditions. The first is a cry for HELP1, and in that case the computer goes to line 440, where it prints the first letter of the word (which the computer carries in its memory as the variable W$), and then offers another go.

Notice how this is done: IF G$="HELP1" THEN 440. At line 440 is printed: LEFT$(W$,1) (that is the first letter from the left), and a penalty is given: WR=WR+2: GOTO 480. At 480 the penalty is added to the WRONG SCORE, followed by GOTO 250, which is the INPUT "Give your Guess". A similar thing happens with HELP2 and HELP3 (with a bigger penalty each time). In the case of line 300 ("GIVEUP") the whole word is printed, and this time the GOTO is to 360 – where a new game is offered.

Those four lines (270 to 300) deal with requests for help. The next two lines (310–320) compare your guess with the correct words, and respond accordingly; either going to 330 (WRONG) and back to 250; or to 340 (CORRECT) and then giving the SCORE before offering another game. It is fairly easy to trace through each sequence of lines for each of the six statements, so make sure you understand exactly what happens in each case.

It will be seen that all the lines from 330 to 480 are subject to the six IF lines, and they give a fair representation of how such matters are arranged during any game: The IF statements sit together, and direct the computer to small

routines elsewhere. One point about the arrangement of the IF statements is important, however: their sequence. If line 310 occurred before the other IF statements the HELP statements would not work: IF G$ (is not equal to) W$, could also apply to HELP1 etc. Therefore watch the sequence of IF lines to see that an all-embracing line (such as line 310) comes after conditional lines like 270–300.

Notice that in lines 380–390 both upper and lower case letters are allowed for. This is in line with the entry mentioned earlier (line 260 G$=UPPER$(G$)). A great number of errors in programs are caused by being in lower case when upper case is called for, and vice versa. In my programs I usually stick to CAPS MODE, and if an error occurs in working I first check to see that I have not inadvertently reverted to lower case before looking for more serious trouble. Wherever possible use entries that will obviate such problems, such as lines 260, 380, and 390.

Go over lines 230 to 480 once again, noting how the WINDOW hash numbers have been used. Both hash 1 and hash 2 are used in lines 330 and 340. Lines 60 to 350 contain all the procedures for WINDOW, and could be saved to a Printer.

The small routine in lines 640 to 680, coupled with DATA (at 800 onward) is an example of how to READ DATA at RANDOM, so take a copy of that on the Printer, too.

```
10 REM ****************************
20 REM ***** GUESS THE WORD *****
30 REM ****************************
40 INK 0,23:INK 1,9:INK 2,2:INK 3,6
50 BORDER 23:PAPER 0:PEN 1:CLS
60 WINDOW #1,22,40,6,12:PEN #1,2
70 WINDOW #2,1,21,4,25
80 PEN 3:LOCATE 2,2:PRINT "*** GUESS
 THIS WORD ";CHR$(243):PEN 1
90 ORIGIN 376,360:DRAW 256,0,2:DRAWR
 0,32:DRAWR -256,0:DRAWR 0,-32
```

```
100 LOCATE #1,5,1:PRINT #1,"SCORE"
110 LOCATE #1,1,3:PRINT #1,"WRONG:":
WR=0:LOCATE #1,9,3:PRINT #1,WR
120 LOCATE #1,1,5:PRINT #1,"RIGHT:":
CO=0:LOCATE #1,9,5:PRINT #1,CO
130 LOCATE #1,1,7:PRINT #1,"PROPORTI
ON:"
140 LOCATE 22,14:PRINT "ENTER 'HELP1
":LOCATE 24,15:PRINT "or HELP2":LOCA
TE 24,16:PRINT "or HELP3":LOCATE 24,
17:PRINT "for letters"
150 LOCATE 22,19:PRINT "GIVEUP for E
ND"
200 REM **********************
210 REM ******* GAME *******
220 REM **********************
230 GOSUB 600
240 PRINT #2
250 PRINT #2," GIVE YOUR GUESS"
260 INPUT #2,G$:G$=UPPER$(G$)
270 IF G$="HELP1" THEN 440
280 IF G$="HELP2" THEN 450
290 IF G$="HELP3" THEN 460
300 IF G$="GIVEUP" THEN 470
310 IF G$<>W$ THEN 330
320 IF G$=W$ THEN 340
330 PEN 3:PRINT #2,"     WRONG":PEN 1
:WR=WR+1:LOCATE #1,9,3:PRINT #1,WR:G
OTO 250
340 PEN 3:PRINT #2,"     CORRECT":PEN
 1:CO=CO+1:LOCATE #1,9,5:PRINT #1,CO
350 LOCATE #1,11,7:PRINT #1,CO;"/";W
R
360 PEN 3:LOCATE 24,22:PRINT "TRY AG
AIN(Y/N)":PEN 1
370 LOCATE 24,24:INPUT T$
```

```
380 IF T$="Y" OR T$="y" THEN 400
390 IF T$="N" OR T$="n" THEN 410
400 CLS:RUN
410 CLS:LOCATE 10,12:PRINT "THANK YO
U FOR THE GAME."
420 GOTO 420
430 END
440 LOCATE 22,21:PRINT LEFT$(W$,1):W
R=WR+2:GOTO 480
450 LOCATE 22,21:PRINT LEFT$(W$,2):W
R=WR+4:GOTO 480
460 LOCATE 22,21:PRINT LEFT$(W$,3):W
R=WR+6:GOTO 480
470 LOCATE 22,21:PRINT W$:GOTO 360
480 LOCATE #1,9,3:PRINT #1,WR:GOTO 2
50
600 REM **********************
610 REM *** FORM ANAGRAM ***
620 REM **********************
630 DIM J$(LEN(A$))
640 RANDOMIZE TIME
650 B=RND*50
660 FOR C=0 TO B
670 READ W$
680 NEXT C
690 Y=0
700 A$=W$
710 FOR X=1 TO LEN(A$)
720 L=LEN(A$)
730 J$=""
740 N=INT(RND*L)+1
750 J$=MID$(A$,N,1)
760 LOCATE X+25,Y+2:PRINT J$
770 A$=LEFT$(A$,N-1)+RIGHT$(A$,L-N)
780 NEXT
790 RETURN
```

```
800 REM **********************
810 REM ******* DATA *******
820 REM **********************
830 DATA ABANDONED,ABERRATION,ABHORR
ENCE,ABOMINATION,ABSOLUTISM,ACADEMIC
IAN,ACCELERATING,ACCOMMODATION,ADMIN
ISTRATIVE,AGGRESSIVE
840 DATA BACILLARY,BACCHANALIAN,BACT
ERIOPHAGE,BARRACUDA,BEATIFICATION,BE
DRABBLED,BEDEVILMENT,BICARBONATE,BLA
MEWORTHY,BOOMERANG
850 DATA CADAVEROUS,CALCULATION,CAND
ITATURE,CENTENNIAL,CENTRIFUGAL,CEREM
ONIAL,CHAMBERLAIN,CHARACTERIZE,CINEM
ATOGRAPH,CIRCUMPOLAR
860 DATA DAGUERREOTYPE,DECAPITATE,DE
CLINATION,DEDICATION,DEFINITIVE,DELI
BERATION,DIAMETRICAL,DISEMBARRASS,DI
SPROPORTIONATE,DOCTRINAIRE
870 DATA EARTHQUAKE,EBULLIENT,ECCLES
IASTICAL,ECONOMICALLY,EDUCATION,EFFL
UVIUM,EGALITARIAN,EGREGIOUS,ELECTION
EER,ELECTROCUTION
```

Chapter Six

CROSSWORD SOLVER

Anyone interested in word games is usually interested in solving crossword puzzles, and one of the difficulties of that game is the abundance of words that have gaps in them – gaps to which the existing letters seem to give no clue. Then comes the time to use pencil and paper for inserting various letters in those gaps; and what a frustrating business that can be. However, do not despair, for I can provide some help.

In this chapter is a program optimistically called 'CROSSWORD SOLVER', which fills those gaps for you with each letter of the alphabet in turn. That still leaves a great deal of the decision-making to you, however; for example: AFFECT & EFFECT. The clue should help in such instances, of course, although some unkind clues can be very misleading.

The program is very simple, and needs little explanation. After the introductory lines defining INK etc., comes GOSUB 3530, leading to a re-defined CHR$ character (209), of which more later. Then comes RETURN (to 70) which is GOSUB 2000. This leads to a screen of instructions, and then, at the touch of the Space-bar, to the game. Note how that move is made (2070–2090). A plain PRINT statement 'Press any key' is followed by Z$=INKEY$:IF Z$="" THEN 2080. This is a line that stops all movement by the computer until a key is pressed. There is no further 'IF' statement defining a key – just CLS. Therefore, if any key is pressed the computer will proceed, clear the screen, and move on to the next line. In this case it is to the lines in section 3000 that produce the screen display for the game. After a few lines of instruction a long rectangle is drawn, in which the word to be worked on is printed, complete with dashes in place of the missing letters.

Above the rectangle is a line of numbers (1 to 26) which indicate the number of the letter or dash below it. Between the number and the letter (or dash), and intersecting the top line of the rectangle, is a line of divisions printed by that newly defined character 209 (in line 3080). Notice that this is entered as: PRINT STRING$(26,209), and that it is not

necessary to enter CHR$(209), because the second number after STRING$ is understood (by the computer) to be a character number. The first number details how many of that character.

These divisions have been put there to make it easier to see which number applies to each letter or dash. There are 26 spaces for letters, which should be long enough for most words. Line 3100 sends the computer back to 80, which is an INPUT asking you to enter the word that puzzles you, using dashes to represent the gaps. Line 90 makes sure that if you use a lower case letter it is at once converted to a capital letter, and then the computer is held for your entry.

Line 100 prints the word, complete with dashes, in the rectangle, line 110 erases the INPUT question, and 120 provides another INPUT 'Enter space No.', which is given the variable J. Line 130 holds the computer, and 140 gives values to the X,Y coordinates for future use. Lines 150 and 160 erase one INPUT and produce another one, while 170 is GOSUB 1000.

This is the section that prints all the alphabet, one letter at a time, in the space asked for in line 120. Line 1030 defines VS as the alphabet, but in this case with the letters re-arranged: the vowels have been grouped together at the beginning of the string, so that they can be rotated through the space in turn, to be followed by the consonents in their turn.

The mechanism for printing the alphabet in turn is one that has been used previously, but is now altered slightly to give a different effect. Line 1040 gives L the value of the LENgth of VS; 1050 is the beginning of a FOR/NEXT loop of N=1 to L; and 1070 PRINTs MID$(VS,N,1): that is one letter from VS selected in turn by N on each pass of the loop. Line 1080 is a pause (to be explained later); and 1090 is a line in which the pressing of the Space-bar (No. 47) will cause the rotation of the letters to stop. Then the computer returns to line 180 – ELSE if it is not pressed the computer continues with the pause to the full time of '1 to 1000'.

The length of this pause will not suit everyone, and it can be altered. Change it to 100, and the letters will flick through so quickly that it will be hard to press the Space-bar in time

to catch the letter you want. It is up to you when the program is entered and running to alter it to suit yourself. I find I need plenty of time to consider each letter in turn.

When the Space-bar is pressed, the letter at present in the space will remain there, and the computer will print the INPUT 'Enter space No.' once again. You can then go over the same space again, or any other space you wish, and this procedure can be repeated as often as you wish.

This is the last of the small programs used in this book to demonstrate in detail the various ways of handling letters and words, for the remaining chapters are devoted to writing larger programs from the routines used previously. It will be as well to go over those routines once again.

In Chapter One anagrams are produced from any entered word. Ten anagrams are made, but this could be any number, just by altering the Q value in line 2050: 1 to 100 would produce 100 anagrams of the word.

Chapter Two uses the shuffling routine in a different way to produce what is (in effect) an anagram of the alphabet, and includes a line (620) to ensure that no letter is used more than once. Inside the shuffling routine is a smaller one that selects locations for placing the letters in a block of five by five; not in a straight line as is more usually done. This is achieved with two FOR/NEXT loops: one for Y and one for X (lines 550–560), with a set of conversions at section 2000 onward.

Chapter Three approaches the problem of word and letter handling in a different way. This time it is to put given words into alphabetical order. Two programs are incorporated into one here: the first to accept the words entered one by one; and the second to select the words from a DATA statement (this one is used by me to provide an index for this book). This is the action of putting into order, instead of shuffling. In addition, a way of using WINDOW is shown. The two programs can be used separately: RUN will produce the first program; and RUN 3000 will produce the second one.

In Chapter Four is a program that once again shuffles the letters of the alphabet, but it also shuffles their positions on the screen, and prints them randomly in any of the three colours available in MODE 1. These three random routines are all incorporated into lines 200–370. In addition a ball is

moved around the screen with the cursor arrows, and the way to do this is shown in lines 400–590.

Also in Chapter Four is another program 'How Many Words' which begins to gather together previously demonstrated routines. This time, however, it is the programmer who does most of the thinking. The way the game routine (lines 1000–1220) has been arranged is worthy of note in this instance – as well as the use of some string variables.

In Chapter Five is a small game using the anagram routine from Chapter One, but it is most notable for its use of two WINDOWs, and how they have been arranged. It is possible with their aid to be able to enter very many words, and to have them scroll up the screen without disturbing the rest of the screen layout. This can be most useful in many games – and utilities.

This review of some of the previous routines has been given here in an abridged form as a reference section for you to be able to find them easily when working out a routine to suit a program you may have in mind. The following chapters are all of complete games; except for the final chapter, which deals with producing a copy of the screen display on the Printer.

```
10 REM ************************
20 REM *** CROSSWORD SOLVER ***
30 REM ************************
40 INK 0,26:INK 1,1:INK 2,2:INK 3,6
50 BORDER 15:PAPER 0:PEN 1:CLS
60 GOSUB 3530
70 GOSUB 2000
80 LOCATE 2,23:INPUT "ENTER SPACES &
 LETTERS";W$
90 W$=UPPER$(W$):IF W$="" THEN 90
100 LOCATE 4,20:PRINT W$
110 LOCATE 2,23:PRINT SPC(38)
120 LOCATE 2,23:INPUT "Enter space N
O.";J
```

```
130 IF J=0 THEN 130
140 Y=20:X=3+J
150 LOCATE 2,23:PRINT SPC(38)
160 LOCATE 2,23:PRINT "USE SPACE-BAR
 TO FIX"
170 GOSUB 1000
180 GOTO 110
1000 REM ************************
1010 REM *** ROTATE LETTERS ***
1020 REM ************************
1030 V$="AEIOUBCDFGHJKLMNPQRSTVWXYZ"
1040 L=LEN(V$)
1050 FOR N=1 TO L
1060 LOCATE X,Y
1070 PRINT MID$(V$,N,1)
1080 FOR Q=1 TO 1000
1090 IF INKEY(47)=0 THEN RETURN ELSE
 NEXT Q
1100 NEXT N:RETURN
2000 REM ************************
2010 REM ***** INFORMATION *****
2020 REM ************************
2030 PEN 3:LOCATE 6,2:PRINT " * * SO
LVING  CROSSWORDS * *":PEN 1
2040 PRINT:PRINT "   When workin9 on
 crosswords one often  becomes stuck
 with 9aPs in a word.      This Pro9
ram can helP by rotatin9 the   alPha
bet throu9h those 9aPs."
2050 PRINT:PRINT "   Enter the word
comPlete with its      sPaces (repre
sented by dashes) and      the alPha
bet can be rotated throu9h     each
of those sPaces in turn."
2060 PRINT:PRINT "   It is Possible
to 9o back over any    letter that d
```

```
oes not fit, and the       Program c
an be used until satisfied     with
the result."
2070 PRINT:PRINT "   Press any key t
o start."
2080 Z$=INKEY$:IF Z$="" THEN 2080
2090 CLS
3000 REM ********************
3010 REM ***** SCREEN *****
3020 REM ********************
3030 PEN 3:LOCATE 11,2:PRINT "CROSSW
ORD  SOLVER":PEN 1
3040 PRINT:PRINT "   Enter letters a
vailable in word form, using a dash
(-) to indicate each       space (li
ke this: -S-I--T- )."
3050 PRINT:PRINT "    Indicate space
 to try out letters    by the number
 above the dash."
3060 PRINT:PRINT "    Respond to the
 questions as they     appear.  Fix
letter in position by      using the
 Space-bar.  You can go back   over
any space."
3070 LOCATE 4,18:PRINT "123456789012
3456789012345678":LOCATE 13,17:PRINT "
1111111111222222"
3080 PEN 2:LOCATE 4,19:PRINT STRING$
(26,209):PEN 1
3090 ORIGIN 40,72:DRAW 436,0,2:DRAWR
 0,32:DRAWR -436,0:DRAWR 0,-32
3100 RETURN
3500 REM ********************
3510 REM ***** U.D.Gs *****
3520 REM ********************
3530 SYMBOL AFTER 200
```

```
3540 SYMBOL 209,1,1,1,1,1,1,1,1
3550 RETURN
```

Chapter Seven

CROSS-JOIN WORDS

The game of 'CROSS-JOIN WORDS' is for one player, who has to use his intelligence and knowledge of words to form part of a crossword puzzle. Seventeen letters are given, and from these a word should be constructed (of any length) to fit somewhere on the board – preferably starting near the centre, which is marked with a star. The second, and following words should incorporate a letter from an existing word as well as some of the seventeen shown freshly after each entry. They are entered either across to the right, or down from a point. This starting point of the word is asked for, and must be given accurately enough to see that the letter incorporated from another word is used in its proper place.

The listing (given at this chapter end) consists of six sections, and follows a format similar to that recommended for previous programs. After the INKs and BORDER have been defined is GOSUB 4000, which leads to a screenful of instructions, including a demonstration of some cross-joined words in a small square. At the bottom of the screen is: 'Press G for the game'. Notice how a quick change-over is arranged with four lines. PRINT is used instead of INPUT; K$ is made to equal INKEY$; and then comes: IF K$ = " " . This holds the computer. The next line prevents an error of capital or lower case letter being used, and 4140 is CLS:GOTO 5000. That GOTO 5000 could have been left out, for in any case the computer would move on to the next lines when once a letter is pressed.

In section 5000 (the game board) are some brief playing instructions, and a board is drawn to contain the words. Location letters are printed at the top and left-hand side of the board, and there is a long rectangle below it to contain the seventeen letters for each word. Then comes RETURN, which sends the computer back to the line after GOSUB 4000, which is GOSUB 2000.

This is the section for selecting the seventeen letters to print in the rectangle, and consists of the shuffle routine

used in previous chapters, but with the addition of a small routine that prints each letter with a space after it. 2030 makes A$ equal to all the letters of the alphabet, and 2040 is RANDOMIZE TIME, so that the randomly selected letters start at a different letter each time. Then comes the new FOR line that is used to space out the letters: FOR S=1 TO 34 STEP 2. This S is used in line 2090 to locate the printing of the shuffled letters. The X location to start with is 3 + S, so that an extra 2 is added to X on each pass through the loop. Line 2100 is the standard line that was mentioned previously (in earlier programs) that extracts the chosen letter from A$, so that in the next pass through, A$ is one letter less. This prevents any letter repetition.

In the case of this game it is doubtful whether it is an advantage to prevent letter repetition, and if you decide that you would prefer to have some letters repeated at times, then all you need do is place a REM at the beginning of line 2100. The routine works just as well without this line. After all the seventeen letters have been printed the computer goes back to the line after GOSUB 2000.

This is to the Game section, where the moves are made. Line 1040 is an INPUT asking you to enter a word. Line 1050 holds the computer until ENTER is pressed, so there is plenty of time to correct or alter the word before it is entered. Line 1060 makes sure that if you enter lower case letters by mistake they are corrected to capitals.

When it is entered line 1070 erases the question, and 1080 prints another one asking for the location. There are three of these location questions: the top letter; the side letter; and whether the word is to be printed horizontally (A), or vertically (D). These are PRINT queries instead of INPUTs, so that they do not require the use of ENTER in order to function.

In answer to the first query (which must be a top letter) the computer is directed to 3000, where are the X conversions. C$ represents the entry, and in lines 3030 to 3170 X is made equal to the number corresponding to the space under the letter entered. 3180 RETURN sends the computer back to the next query, which is for the letter at the side of the board.

This is in line 1120, and following this is a similar routine at line 3190–3330, but for the Y locations this time. D$ is

the variable used this time, and after the Y number has been selected the RETURN in line 3340 sends the computer back to the third query. This one, at 1160, asks you to press A or D (for across or down), and is followed by GOSUB 3530. The variable is P$, and the 'Word Direction' section at 3500–3640 consists of three small sections: the first one (lines 3530–3540) sends the computer to the second or third section, according to whether A or D was pressed.

These two sections are similar, except that one introduces a FOR/NEXT loop into the X location (to print the letters sideways), and the other uses a loop in the Y section to print the word downward. The use of –1 in the location is occasioned by the use of 1 TO M in the FOR/NEXT loop. If 0 TO M was used, then the print statement would be in error in line 3570, for G cannot (in that context) represent 0 TO M. Therefore X=G would be 1 space to the right of the correct position. This applies similarly to the Y position.

The word is now printed in its proper place, and the appropriate RETURN (from 3590 or 3640) sends the computer back to line 1190, where the query is erased; then to 1200, which sends the computer back to 1030; where the routine starts again with the printing of seventeen different letters in the rectangle, followed by the INPUT query: "Enter Word".

There are a number of points throughout this program that are worth bearing in mind, for they can be useful in many other programs. Take lines 1070, 1110, 1150, and 1190: these merely print a row of spaces, thus erasing the previous query to make room for the next one. They are not essential, for the following query in that same place would erase the first one – except when the previous question was longer than the second, for then some letters would remain to confuse the sense of the following question. This method of erasing is the best way to do that job.

Notice the different ways of phrasing an INPUT query and a PRINT query (Press only). The INPUT question is followed by a semicolon (;), and then by a string variable. The PRINT query has nothing after it in the same line, but in the next line is always: C$ (or any other variable) = INKEY$, and then the IF (variable) = " " line. Grasp this principle thoroughly,

and all kinds of input become very much easier to arrange.

In line 2040 is RANDOMIZE TIME. Without that extra line RND would produce a form of random number that was not truly random, for it is worked out by the computer starting from the same base each time, and must inevitably repeat sometimes. Using RANDOMIZE TIME ensures that RND is more truly random, for it works from a number coupled to the length of time since the computer was switched on. It isn't always necessary to use RANDOMIZE TIME, but often it is essential.

There is a use of the keywords TAG and TAGOFF that has not been mentioned in this book. When this program is entered and running correctly, EDIT line 5050, and insert REM at its beginning to prevent it functioning. Then enter (as an extra line): 5055 TAG:ORIGIN 56,352:PEN 3:PRINT "ABCDEFGHIJKLMNO";:PEN 1:TAGOFF:LOCATE 22,4: PRINT "K (5 spaces here) ARE GIVEN".

This is actually line 5050 altered, and with the addition of TAG and TAGOFF; the first LOCATE 4,4 has been altered to 'ORIGIN 56,352'. This is 8 pixels to the right of the original position. By this means text or graphic characters can be put in other than the normal character spaces.

If the program is now RUN it will be seen that the top numbers above the board game have been moved slightly to the right, so that they now sit over the lines, instead of over the spaces. This alteration is definitely not wanted here, but there are many times when it would be an advantage to be able to do this. Make sure you really understand how this works – but remember one extra point: the characters being printed inside a TAG/TAGOFF statement must always be followed by a semicolon (;). If you omit this some peculiar characters will be printed after the line of text.

For those of you who have a DMP1 Printer, and who wish to print out just one line from a program, use this formula: LIST 5055–5055 #8. That will do the trick. Of course, you will use the appropriate line number. Similarly, using numbers like this (5030–5090) will select just those lines from the program to print.

```
10 REM ************************
20 REM *** CROSSJOIN WORDS ***
30 REM ************************
40 INK 0,26:INK 1,1:INK 2,2:INK 3,6
50 BORDER 8:PAPER 0:PEN 1:CLS
60 GOSUB 4000
1000 REM ********************
1010 REM ******* GAME *******
1020 REM ********************
1030 GOSUB 2000
1040 LOCATE 4,23:INPUT "ENTER WORD";
W$
1050 IF W$="" THEN 1050
1060 W$=UPPER$(W$)
1070 LOCATE 4,23:PRINT SPC(30)
1080 PEN 3:LOCATE 4,23:PRINT "PRESS
TOP LETTER":PEN 1
1090 C$=INKEY$:IF C$="" THEN 1090
1100 GOSUB 3000
1110 LOCATE 4,23:PRINT SPC(30)
1120 PEN 2:LOCATE 4,23:PRINT "PRESS
SIDE LETTER":PEN 1
1130 D$=INKEY$:IF D$="" THEN 1130
1140 GOSUB 3190
1150 LOCATE 4,23:PRINT SPC(30)
1160 LOCATE 4,23:PRINT "PRESS A OR D
"
1170 P$=INKEY$:IF P$="" THEN 1170
1180 GOSUB 3530
1190 LOCATE 4,23:PRINT SPC(30)
1200 GOTO 1030
2000 REM ************************
2010 REM *** SELECT LETTERS ***
2020 REM ************************
2030 A$="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
2040 RANDOMIZE TIME
```

```
2050 FOR S=1 TO 34 STEP 2
2060 J$="":L=LEN(A$)
2070 R=INT(RND*L)+1
2080 J$=MID$(A$,R,1)
2090 LOCATE 3+S,21:PRINT J$
2100 A$=LEFT$(A$,R-1)+RIGHT$(A$,L-R)
2110 NEXT S:RETURN
3000 REM ********************
3010 REM *** CONVERSIONS ***
3020 REM ********************
3030 IF C$="A" OR C$="a" THEN X=4
3040 IF C$="B" OR C$="b" THEN X=5
3050 IF C$="C" OR C$="c" THEN X=6
3060 IF C$="D" OR C$="d" THEN X=7
3070 IF C$="E" OR C$="e" THEN X=8
3080 IF C$="F" OR C$="f" THEN X=9
3090 IF C$="G" OR C$="g" THEN X=10
3100 IF C$="H" OR C$="h" THEN X=11
3110 IF C$="I" OR C$="i" THEN X=12
3120 IF C$="J" OR C$="j" THEN X=13
3130 IF C$="K" OR C$="k" THEN X=14
3140 IF C$="L" OR C$="l" THEN X=15
3150 IF C$="M" OR C$="m" THEN X=16
3160 IF C$="N" OR C$="n" THEN X=17
3170 IF C$="O" OR C$="o" THEN X=18
3180 RETURN
3190 IF D$="A" OR D$="a" THEN Y=5
3200 IF D$="B" OR D$="b" THEN Y=6
3210 IF D$="C" OR D$="c" THEN Y=7
3220 IF D$="D" OR D$="d" THEN Y=8
3230 IF D$="E" OR D$="e" THEN Y=9
3240 IF D$="F" OR D$="f" THEN Y=10
3250 IF D$="G" OR D$="g" THEN Y=11
3260 IF D$="H" OR D$="h" THEN Y=12
3270 IF D$="I" OR D$="i" THEN Y=13
3280 IF D$="J" OR D$="j" THEN Y=14
```

```
3290 IF D$="K" OR D$="k" THEN Y=15
3300 IF D$="L" OR D$="l" THEN Y=16
3310 IF D$="M" OR D$="m" THEN Y=17
3320 IF D$="N" OR D$="n" THEN Y=18
3330 IF D$="O" OR D$="o" THEN Y=19
3340 RETURN
3500 REM ********************
3510 REM * WORD DIRECTION *
3520 REM ********************
3530 IF P$="A" OR P$="a" THEN 3550
3540 IF P$="D" OR P$="d" THEN 3600
3550 M=LEN(W$):FOR G=1 TO M
3560 LOCATE X-1+G,Y
3570 PRINT MID$(W$,G,1)
3580 NEXT G
3590 RETURN
3600 M=LEN(W$):FOR H=1 TO M
3610 LOCATE X,Y-1+H
3620 PRINT MID$(W$,H,1)
3630 NEXT H
3640 RETURN
4000 REM ************************
4010 REM ***** INSTRUCTIONS *****
4020 REM ************************
4030 PEN 3:LOCATE 13,2:PRINT "CROSSJ
OIN WORDS":PEN 1
4040 LOCATE 3,4:PRINT "A   D":PRINT
" AVENUE":PRINT "  E   F":PRINT "  R
   I":PRINT "  AGRONOMY":PRINT "EGGS
  I":PRINT "  EIGHTEEN":PRINT "
 E"
4050 FOR X=0 TO 168 STEP 16:Y=224:OR
IGIN X,Y:DRAW 0,128,2:NEXT X:FOR Y=2
24 TO 360 STEP 16:X=0:ORIGIN X,Y:DRA
W 160,0:NEXT Y
4060 LOCATE 12,4:PRINT "  The aim is
```

```
 to form words":LOCATE 12,5:PRINT "f
rom the 17 letters":LOCATE 12,6:PRIN
T "already on the board.":LOCATE 12,
8:PRINT "  Words can be entered acro
ss"
4070 LOCATE 12,9:PRINT "or down, usi
ng the top & side":LOCATE 12,10:PRIN
T "letters to place the first":LOCAT
E 12,11:PRINT "letter of the word--n
ear the":LOCATE 12,12:PRINT "board c
entre is best."
4080 LOCATE 2,13:PRINT "  Each word
entered after the first wordmust use
 one or more of the letters of  word
s already on the board."
4090 LOCATE 2,17:PRINT "  When the b
oard is nearly full the     pattern
of words could be used for an   orig
inal Crossword Puzzle.  It would
just require black squares to fill t
he  spaces, and some misleading clue
s."
4100 ORIGIN 0,0:DRAW 639,0,2:DRAWR 0
,399:DRAWR -639,0:DRAWR 0,-399
4110 PEN 3:LOCATE 2,24:PRINT "Press
G for the game":PEN 1
4120 K$=INKEY$:IF K$="" THEN 4120
4130 IF K$="G" OR K$="g" THEN 4140
4140 CLS:GOTO 5000
5000 REM ************************
5010 REM ***** GAME BOARD *****
5020 REM ************************
5030 LOCATE 4,2:PEN 3:PRINT "CROSSJO
IN WORDS   ":PEN 1:LOCATE 22,2:PRINT
 "L    17   LETTERS"
5040 LOCATE 20,3:PRINT "TH":PEN 3:LO
```

```
CATE 22,3:PRINT "I":PEN 1:LOCATE 23,
3:PRINT "S"
5050 LOCATE 4,4:PEN 3:PRINT "ABCDEFG
HIJKLMNO":PEN 1:LOCATE 22,4:PRINT "K
     ARE GIVEN"
5060 PEN 2:LOCATE 1,5:PRINT "  A":PR
INT"  B":PRINT"  C":PRINT"  D":PRINT
"  E":PRINT"  F":PRINT"  G":PRINT"
H":PRINT"  I":PRINT"  J":PRINT"  K":
PRINT"  L":PRINT"  M":PRINT"  N":PRI
NT"  O":PEN 1
5070 LOCATE 22,5:PRINT"E":LOCATE 24,
6:PRINT "Use any of these":LOCATE 24
,7:PRINT "letters to make":LOCATE 24
,8:PRINT "a word, & then"
5080 LOCATE 22,10:PRINT "ENTER IT CO
MPLETE."
5090 LOCATE 22,12:PRINT "Then locate
 Place":LOCATE 22,13:PRINT "for 1st
letter,":LOCATE 23,14:PRINT "& then:
_"
5100 LOCATE 23,16:PRINT "PRESS ToP l
etter.":LOCATE 23,17:PRINT "PRESS si
de letter.":LOCATE 23,18:PRINT "PRES
S A(across)":LOCATE 26,19:PRINT "or
D(down)"
5110 FOR X=48 TO 288 STEP 16:Y=96:OR
IGIN X,Y:DRAW 0,240,1:NEXT X
5120 X=48:FOR Y=96 TO 336 STEP 16:OR
IGIN X,Y:DRAW 240,0,1:NEXT Y
5130 PEN 2:LOCATE 11,12:PRINT "*":PE
N 1
5140 ORIGIN 40,56:DRAW 544,0,2:DRAWR
 0,32:DRAWR -544,0:DRAWR 0,-32
5150 ORIGIN 0,0:DRAW 639,0,2:DRAWR 0
,399:DRAWR -639,0:DRAWR 0,-399
5160 RETURN
```

Chapter Eight

THINKING OF A WORD SQUARE

The game used in this chapter is one that has been about in various forms for many years, but it always arouses interest. It consists of entering a word (of any length up to nine letters in this version), and then thinking of another word of the same length that is different, but that uses the last letter of the first word; it is entered directly below the first. The third word must again be different, but must use the last two letters of the second word. All these repeated letters must fall directly below the one above. This goes on until a complete square of words has been formed; as many words as there are letters in the first word. An example is shown at the end of the chapter.

When using words that are four or five letters long it is relatively easy to complete a square of words, but when it comes to words of eight or nine letters each it is not so easy. This is a game that stretches minds considerably, and is designed for a single player, or for two players; each one supplying alternate words.

There are five main sections in the program: the initialisation of the colours; the game moves; a section for printing the moves; an example of how the game is played; and the screen display. After the colours have been defined, the program starts at line 60; GOSUB 3000.

It may be helpful to describe here why line 60 is followed by the Game section at 1000, placing the Example and Screen sections at the program end. Every time the computer is given an instruction (such as one of the many GOSUB commands used in the Game section), it searches through all the line numbers from the beginning to find the line wanted. If the Example section at 3000 and the Screen display section at 4000 were placed between line 50 and the Game section, the computer would have to race through those every time it came to a command. Placing these two sections at the end saves playing time. Similarly, to save time when playing, the various GOSUB lines used during play are to a section at 2000,

immediately after the Game section.

Therefore line 60 GOSUB 3000 points to a section that is used only once at the start of the game, and is then ignored. In 3000 a square of words is shown, together with a few instructions, using ORIGIN/DRAW and LOCATE/PRINT statements. This instruction section is extra to the game itself, and, after you understand the game, could well be deleted. However, it is included because if the game has not been played for some time, the reminder is worth seeing. At this section end, in lines 3100–3120, is a routine to move on to the next section. This is arranged so that pressing letter G erases the example, and produces the screen display. Notice how it is done with PRINT instead of INPUT in line 3100; a 'hold' statement in 3110; and the action statement in 3120. This also contains a precaution against the error of using a lower case letter. Line 3120 erases the example, and the computer then moves on to the next lines, to section 4000, where is the Screen display.

Here is a similar square-drawing sequence (to contain the words), and the playing instructions are stated briefly. At the end of line 4090 is a RETURN, which sends the computer back to the Game section (to the line after GOSUB 3000) without erasing the screen display. From then on throughout the game the computer has no need to scan lines 3000 onward.

At line 1000 the game starts; line 1030 is an INPUT asking for the first word to be entered. This is given the variable A$, and each of the words afterwards are given a different variable (B$, C$, D$, etc.) up to a total of nine words. Line 1050 makes W$ equal to A$. This is done so that a single routine (at 2000) can handle all the words in turn (B$ and then the other variables for more words will be made equal to W$). X and Y are given a value, and then comes line 1060 GOSUB 2000.

In this small routine (lines 2030–2080) L is given the value of the LENgth of W$, and at 2040 a FOR/NEXT loop starts: FOR N=1 TO L (the length of W$). In lines 2050 and 2060 the first letter of W$ is printed; in 2070 X is given the value of X+2; and in 2080 comes NEXT N, so that the computer goes back to 2040 to print the next letter – 2 spaces to the right. This continues until the length of W$ is used up,

and then comes RETURN, to the line after GOSUB 2000, which in this case is 1070.

There a second routine is started to print in blue ink the end letter(s) which must be used in the next word. In 1070 X is made equal to X–2 so that the blue letter(s) will be printed on top of the black one. 1080 makes WW$ equal to the right-hand letter (in this case) of W$. 1090 is GOSUB 2100.

At 2100 is the small routine to print the 'carried-on' letter or letters. LL is made equal to the LENgth of WW$ (which is just one letter in this first instance, but becomes more with each word entered). In 2110 another FOR/NEXT loop is started using the variable NN (which is: 1 TO LL). At 2120, using PEN 2 (blue), the letter is printed; and at 2130 X is made equal to X+2 (to print the next letter two spaces to the right). NEXT NN repeats the loop if necessary (but not in the case of the first word).

The phrase 'IF LL = L THEN 1710' is inserted at 2130 to ensure that if words of less than nine letters are used (I recommend that you try first with words of four or five letters) then when the last word is entered (when the blue letters equal the length of W$) the end routine at 1710 is used.

This comparison of LL with L shows how well the Amstrad 464, 664 or 6128 can cope with similar-looking variables: NN and N, LL and L, and even W$ and WW$ are all treated as distinctly different. In this program the string variables (those with $ that can handle words) include A to I and K, plus W and WW. Ordinary variables (those used for numbers, without $) include X, Y, L, LL, N, and NN.

To become familiar with this reliable facility do this: after the program is all entered, and you have RUN one game, ENTER ESC, for Break, and then enter directly: PRINT B$;" ";D$;" ";WW$. The result will be three words: the second word you entered in the game, the fourth word, and the last word. This shows how well the variables are stored in memory until they are changed by you or by the program. In this program the two string variables W$ and WW$ are continually being changed as the game progresses; try PRINTing them at different stages of the game.

To return to the program description: at 2140 is a

LOCATE/PRINT statement of spaces to erase the previous INPUT statement, and at line 2150 a RETURN sends the computer to line 1110 (in the case of this first word). At 1110 comes the INPUT for the second word, followed by lines almost the same as those at 1030–1090, except that the X and Y numbers are different, and so is line 1160, for now WW$ is made equal to two letters at the right-hand end of the second word.

There are nine small routines between 1030 and 1700, and they are all nearly identical, apart from those differences mentioned earlier. In the ninth word routine, however, there is an extra difference, for in this one instance (at line 1700) the computer is ordered to use PEN 2 (blue) before being sent to 2000 to print the word, and put back into PEN 1 on its RETURN. Also the previously used routine at 2100 is not now needed, so these three lines are omitted. When the computer returns from the routine at 2000 it goes on then to use the finishing lines at 1710 to 1730.

This program has been explained in detail because it is a suitable one for improvements you can make yourself. For instance, one that has been carried out already is this: when the game was first written the end (lines 1710 onward) only appeared at the end of a nine-letter game. If less than nine letters were used the game didn't end, but asked for the next word – even when the final word was already printed in blue, and the square was complete. I altered this by adding just one small phrase at the end of line 2130: IF LL=L THEN 1710. This meant that when LL (the length of the letters to be printed in blue) was equal to the length of L (which was W$) then the game was to end. This sort of improvement as you write a program (but more often when it is finished) makes all the difference.

One alteration you should be able to make now (if you have followed all the chapters in this book) is an addition to line 1720 offering another game. Try working it out. It will entail separating sections 3000 and 4000, with a RETURN after the CLS in line 3120, and an additional line 70 saying GOSUB 4000. Then, in the section at 1710 could be added some lines to clear the screen, and go straight to 4000 for a new start.

When writing almost any kind of program the problems are usually separate ones, and an experienced programmer is one who knows many kinds of tips. For instance, the ability to place a character (letter or number) at a place between the character spaces is easily learnt. It can be done (as explained earlier in this book) by using ORIGIN instead of LOCATE, and placing TAG at the beginning of the line, and TAGOFF at its end. Those two keywords enable you to use ORIGIN 240,222 instead of LOCATE 16,22, which are both exactly the same location. Always remember to use a semicolon (;) after the PRINT statement and before the colon (:) when using ORIGIN for PRINT statements.

In this present program are some tips, and one of them is a way of spacing out the letters of a word. Look at lines 2030 to 2080. When the program is completely entered – and taped – then alter the +2 into +1 in line 2070, and then RUN the program. The letters of each word will be printed close to each other, and the blue letter will appear on the next to last letter; the spacing has been lost. That little routine can be most useful at times. First of all give the word a string variable, then use lines 2030 to 2080 (without the RETURN, of course); giving the X and Y locations whichever value you like.

Now, as an additional facility use TAG before the routine, TAGOFF after it, and ORIGIN numbers instead of LOCATE – not forgetting that semicolon. In that way you can space the letters half a space apart (using X=X+24), or one and a half spaces (using X=X+40). You do realise, of course, that LOCATE numbers run at X from 1 to 40, and at Y from 1 to 25 downward, while ORIGIN numbers run at X from 0 to 639, and at Y from 0 to 399 upward. Another point to remember is that the ORIGIN number must be the one at the top of the character, not at its bottom; there is a matter of 16 pixels between these two points. These are the sort of points to look for when entering other people's programs. One picks up a tip here, and another there, and together they build up quite a lot of experience.

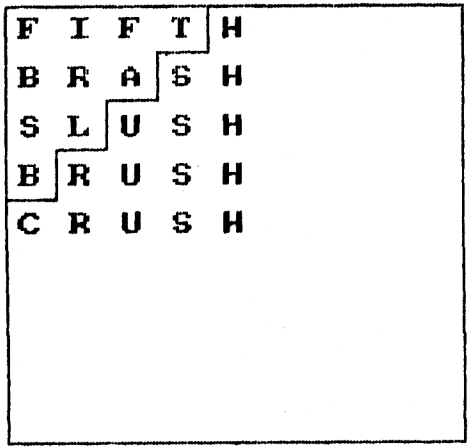
```
10 REM ************************
20 REM *** THINKING OF WORDS ***
30 REM ************************
40 INK 0,25:INK 1,9:INK 2,11:INK 3,6
50 BORDER 2:PAPER 0:PEN 1:CLS
60 GOSUB 3000
1000 REM ****************
1010 REM ***** GAME *****
1020 REM ****************
1030 LOCATE 2,22:INPUT "ENTER A WORD
(to 9 lets.)";A$
1040 IF A$="" THEN 1040
1050 W$=A$:X=2:Y=2
1060 GOSUB 2000
1070 X=X-2
1080 WW$=RIGHT$(W$,1)
1090 GOSUB 2100
1100 REM ***** 2nd. WORD *****
1110 LOCATE 2,22:INPUT "ENTER 2nd. W
ORD";B$
1120 IF B$="" THEN 1120
1130 W$=B$:X=2:Y=4
1140 GOSUB 2000
1150 X=X-4
1160 WW$=RIGHT$(W$,2)
1170 GOSUB 2100
1180 REM ***** 3rd. WORD *****
1190 LOCATE 2,22:INPUT "ENTER 3rd. W
ORD";C$
1200 IF C$="" THEN 1200
1210 W$=C$:X=2:Y=6
1220 GOSUB 2000
1230 X=X-6
1240 WW$=RIGHT$(W$,3)
1250 GOSUB 2100
1260 REM ***** 4th. WORD *****
```

```
1270 LOCATE 2,22:INPUT "ENTER 4th. W
ORD";D$
1280 IF D$="" THEN 1280
1290 W$=D$:X=2:Y=8
1300 GOSUB 2000
1310 X=X-8
1320 WW$=RIGHT$(W$,4)
1330 GOSUB 2100
1340 REM ***** 5th. WORD *****
1350 LOCATE 2,22:INPUT "ENTER 5th. W
ORD";E$
1360 IF E$="" THEN 1360
1370 W$=E$:X=2:Y=10
1380 GOSUB 2000
1390 X=X-10
1400 WW$=RIGHT$(W$,5)
1410 GOSUB 2100
1420 REM ***** 6th. WORD *****
1430 LOCATE 2,22:INPUT "ENTER 6th. W
ORD";F$
1440 IF F$="" THEN 1440
1450 W$=F$:X=2:Y=12
1460 GOSUB 2000
1470 X=X-12
1480 WW$=RIGHT$(W$,6)
1490 GOSUB 2100
1500 REM ***** 7th. WORD *****
1510 LOCATE 2,22:INPUT "ENTER 7th. W
ORD";G$
1520 IF G$="" THEN 1520
1530 W$=G$:X=2:Y=14
1540 GOSUB 2000
1550 X=X-14
1560 WW$=RIGHT$(W$,7)
1570 GOSUB 2100
1580 REM ***** 8th. WORD *****
```

```
1590 LOCATE 2,22:INPUT "ENTER 8th. W
ORD";H$
1600 IF H$="" THEN 1600
1610 W$=H$:X=2:Y=16
1620 GOSUB 2000
1630 X=X-16
1640 WW$=RIGHT$(W$,8)
1650 GOSUB 2100
1660 REM ***** 9th. WORD *****
1670 LOCATE 2,22:INPUT "ENTER 9th. W
ORD";I$
1680 IF I$="" THEN 1680
1690 W$=I$:X=2:Y=18
1700 PEN 2:GOSUB 2000:PEN 1
1710 LOCATE 2,22:PRINT SPC(35)
1720 PEN 3:LOCATE 2,22:PRINT "LAST W
ORD":PEN 1
1730 GOTO 1730
2000 REM **********************
2010 REM *** PLACING WORDS ***
2020 REM **********************
2030 L=LEN(W$)
2040 FOR N=1 TO L
2050 LOCATE X,Y
2060 PRINT MID$(W$,N,1)
2070 X=X+2
2080 NEXT N:RETURN
2090 REM *** BLUE LETTERS ***
2100 LL=LEN(WW$)
2110 FOR NN=1 TO LL
2120 PEN 2:LOCATE X,Y:PRINT MID$(WW$
,NN,1):PEN 1
2130 X=X+2:NEXT NN:IF LL=L THEN 1710
2140 LOCATE 2,22:PRINT SPC(35)
2150 RETURN
```

```
3000 REM ********************
3010 REM ***** EXAMPLE *****
3020 REM ********************
3030 ORIGIN 8,104:DRAW 288,0,2:DRAWR
 0,288:DRAWR -288,0:DRAWR 0,-288
3040 PEN 3:LOCATE 24,2:PRINT "E X A
M P L E":PEN 1
3050 LOCATE 21,4:PRINT " Enter a wor
d(up to":LOCATE 21,5:PRINT "9 letter
s lon9).":LOCATE 21,6:PRINT " Then a
nother(usin9":LOCATE 21,7:PRINT "sam
e last letter."
3060 LOCATE 21,8:PRINT " Then anothe
r(usin9":LOCATE 21,9:PRINT "two last
 letters.":LOCATE 21,10:PRINT " Then
 another(usin9":LOCATE 21,11:PRINT "
three last letters.":LOCATE 21,12:PR
INT " And carry on."
3070 LOCATE 21,14:PRINT " Notice how
 the":LOCATE 21,15:PRINT "final word
 builds":LOCATE 21,16:PRINT "up as y
ou pro9ress."
3080 LOCATE 2,2:PRINT "F I F T H":LO
CATE 2,4:PRINT "B R A S H":LOCATE 2,
6:PRINT "S L U S H":LOCATE 2,8:PRINT
 "B R U S H":LOCATE 2,10:PRINT "C R
U S H"
3090 ORIGIN 8,232:DRAW 0,32,2:DRAWR
32,0:DRAWR 0,32:DRAWR 32,0:DRAWR 0,3
2:DRAWR 32,0:DRAWR 0,32:DRAWR 32,0:D
RAWR 0,32
3100 LOCATE 2,24:PRINT "Press G for
the 9ame"
3110 K$=INKEY$:IF K$="" THEN 3110
3120 IF K$="G" OR K$="9" THEN CLS
```

```
4000 REM ******************
4010 REM ***** SCREEN *****
4020 REM ******************
4030 ORIGIN 8,104:DRAW 288,0,2:DRAWR
 0,288:DRAWR -288,0:DRAWR 0,-288
4040 PEN 3:LOCATE 23,2:PRINT "THINK
OF A WORD":PEN 1
4050 LOCATE 21,4:PRINT " ENTER a wor
d(up to":LOCATE 21,5:PRINT "9 letter
s long)."
4060 PEN 2:LOCATE 21,6:PRINT " Think
 of another":LOCATE 21,7:PRINT "word
(same length &":LOCATE 21,8:PRINT "s
ame last letter).":LOCATE 21,9:PRINT
 "  ENTER THAT.":PEN 1
4070 PEN 3:LOCATE 21,10:PRINT " Thin
k of another":LOCATE 21,11:PRINT "wo
rd(same length &":LOCATE 21,12:PRINT
 "same 2 last letters)":LOCATE 21,13
:PRINT "  ENTER THAT.":PEN 1
4080 LOCATE 21,14:PRINT " Think of a
nother":LOCATE 21,15:PRINT "word(sam
e length &":LOCATE 21,16:PRINT "same
 3 last letters)":LOCATE 21,17:PRINT
 "  ENTER THAT."
4090 PEN 3:LOCATE 24,19:PRINT "AND S
O ON":PEN 1:RETURN
```



E X A M P L E

Enter a word(up to 9 letters long).
Then another(using same last letter.
Then another(using two last letters.
Then another(using three last letters.
And carry on.

Notice how the final word builds up as you progress.

Press G for the game

Chapter Nine

CANYON CROSSING

The program in this chapter, "CANYON CROSSING", has been used to gather together many of the tips and routines mentioned in previous programs, together with extra hints and ways of doing things on the Amstrad 464, 664 and 6128. Most people will find this game fascinating, for in it you have to guess a word, letter by letter; each guess moves a tight-rope walker a little farther across the canyon, but seven wrong guesses causes him to fall off the rope into the river (from which he can escape to try again).

During the planning of this program there were a number of objectives: a reasonable scene of the canyon and tight-rope; a means of entering each letter in turn into its proper place; and a score that would carry on from one try to the next. The list was written first, so that it could be referred to during the writing of the program; and then the structure or skeleton of the program was written down. Lines 10 onward are for the initialisation; 1000 for the game; 2000 for the small routines; 3000 for the DATA; 5000 for the screen display (leaving space for DATA); and 6000 for the WINDOW.

This last section will be explained first. It was visualised that the game title and the score should remain on the screen from one try to the next, and the simplest way to do this was to use the WINDOW facility to contain these, and keep that WINDOW separate from the game itself. Therefore, after the INKs and BORDER had been defined came GOSUB 6000 (line 60), and that section was written next.

In 6030 the WINDOW extent was defined, together with the INK to be used; notice how this is done: the number of WINDOWs available start at 0, so by defining hash 1 the rest of the screen was left as hash 0 by default. This would mean that there would be no need to use the hash sign (#) for the main screen LOCATE/PRINT statements, and only use hash 1 where that was needed. This WINDOW is defined as just the top three rows right across the screen. The PEN number for

hash 1 is defined as 2. In line 40 it will be seen that INK 2 is defined as 2 colours, so it will flash. However, the two colours chosen are 20 bright cyan, and 23 pastel cyan; so close together in the scale of colours that the flashing will not be too pronounced. The reason for this will be explained when we come to the scene.

Line 6040 prints the game title; notice how the hash sign is used: after LOCATE and also after PRINT. In 6050 two words are printed, and there are 12 spaces between them. This places them correctly to receive the scores. In 6060 and 6070 the wrong scores (WR) and the correct ones (RI) are given variables, each worth 0, and these are printed in place. Line 6080 is RETURN, to the line after 60.

Here, at line 70, is a KEY defining line. When writing a program that contains a WINDOW, and when needing to list the program to inspect or alter something, it is annoying to find that it scrolls only in the WINDOW. Therefore this line is entered. It uses the full-stop in the separate block of numbers. The number of that KEY is 138 (this will be found on page 15 of Appendix 111 of the manual) and I find it the most convenient to use. You may wish to use a different KEY. With this line in place it is merely necessary to press this key (after ESC) to get the list on a full screen. Note the way it is defined: the part between the inverted commas is the instruction.

Line 80 is GOSUB 5000, which is the screen display section, and uses all the screen except for the top three lines. This must be understood thoroughly, for the LOCATE Y numbers now start as 1 where 4 used to be, and go down to only 22. That is: the old number 4 is now number 1 (for Y numbers only). Consequently if you are using a plotting board on which to lay out the screen, it is best to use a board with the Y numbers down the left-hand side numbering: 1, 2, 3, 1, 2, 3, 4, 5 – 22. Then, when you need to use a LOCATE without a hash sign (this will be for most of the screen) the Y numbers can be easily read from the board. This WINDOW change of LOCATE Y positions does not affect the ORIGIN/DRAW statements. These, used to draw the canyon and the river, will still use the entire board numbers. Make sure you understand this thoroughly.

Line 5030 emphasizes this matter, for the default WINDOW is defined there as: 1,40 (for the X numbers) and 4,25 (for the Y numbers 1 to 22). Lines 5040 to 5170 are all ORIGIN/DRAW statements that produce the canyon and river. Line 5090 draws the rectangle around the moves of the game; it uses PEN 1 (as is shown by the command: DRAW –296,0,1: that last 1 being the colour). Lines 5100 to 5160 (using PEN 2) draw the river, and herein lies the need for the flashing colour: to provide some semblance of movement to the river. Line 5170 (using PEN 1) draws the tight-rope. After that come lines 5180 to 5220 detailing LOCATE/PRINT statements for the wording on the right and in the rectangular panels, and lines 5230 and 5240 that draw the surround to the scenery and the moves; notice how the top border is produced: STRING$(40,238). Line 5250 is the RETURN – to the game section at 1000 onward.

This starts at 1030 with RESTORE 3030, and is followed by four lines that select a word (randomly) from the 210 words that are in DATA, at 3000. This is a reservoir of words that, so far, contains 30 words starting with A, followed by 30 each of B, C, and so on to G; that is 210 words in all. Line 1040 B=RND*210 refers to that, and if more words are added to the DATA, then that 210 must be altered to correspond.

I suggest that one improvement you can make easily to this program is the enlargement of the DATA section by including all the letters of the alphabet, making 780 words in all. Of course line 1040 must be altered at the same time, but that is all. One further point about the DATA words in this game: if you do decide to use more words (or different words) make sure you do not add any words that have a letter repeated within them. For instance: LETTER would not be any good, whereas LATER would. This is because in line 2640 K (the number of letters entered) is compared with L (the length of A$) in order to find the completed word. That line would need a different comparison to match the word if two or more letters were entered at one go. There is plenty of room for more words between 3240 and 4990.

Line 1040 makes B equal to a randomly selected word from the 210 in DATA. 1050 makes C equal to the length of that

word; 1060 READ A$ picks up the word and gives it the variable A$; and 1070 NEXT C makes the FOR/NEXT loop (1050 to 1070) repeat until it is all there. That is a perfect little formula for selecting a word, and putting it in memory with the variable A$.

Then comes 1080 (which is the length of A$) and in 1090 –1110 are three lines that print dashes (CHR$(45)) to equal the letters in A$. Note the simple way this is done: 1100 for F = 1 TO L:B$=STRING$(F,45):NEXT F. The variable F is made equal to the number of letters in the word. B$ is made equal to STRING$(length of F) of 45 (which is CHR$(45) from the Amstrad list of characters – a dash). It is not necessary to include the letters CHR$ in such a quotation; STRING$(F,45) is equivalent to STRING$(F,CHR$(45)).

In 1120 a number of variables are given values to be used in the program. Two are used in 1130, where an INPUT asks for a letter, which is given the variable Z$. 1140 makes sure that if you inadvertently use a lower case letter it is converted to a capital letter. Then come two FOR/NEXT loops to compare Z$ with each letter of A$ (the randomly selected word). The first one (at 1160) sends the computer to 1300 if the letter matches one in the word; the second one (line 1190) sends it to 1400 if it does not match. We will examine these in turn.

In line 1300 the computer locates the position to print a correct letter, and prints it there. It is then sent to 2400, which is the section that prints the letter in its correct place (in the row of dashes). 2430 establishes X as 23, and in 2440 INSTR searches A$ for the presence of Z$ (the entered letter), starting at its first letter: IF INSTR(1,A$,Z$) > 0.

The signs ' > 0 ' means more than nothing, or 'IF the letter is present'. This is one of the ways in which computer language is difficult to grasp at first: 'if the presence of the letter is not equal to nothing'. It doesn't always seem to be sensible – but it works. On examination one gets a clue: after all, two negatives are supposed to equal a positive, aren't they? Just worry over it a little, and it begins to make sense.

The next line, 2450, starts a FOR/NEXT loop, using P=1 TO L. Now L is the length of A$ (this was established

in 1080), so P equals each letter of A$ in turn. 2460 locates this particular P + X–1 and Y+5 (that is the particular dash that replaces that letter). 2470 says that if the particular position of P in A$ is equal to the letter entered, then print it there. 2480 makes a SOUND and then comes NEXT P.

In 2490 is K=K+1. Now, in 1120 K was given the value of 0; in 2490 (when a correct letter is printed) the value of K is increased by 1. Then comes GOSUB 2600, where is a routine to establish whether or not the number of K is equal to L (the length of A$). If it is, then the computer goes to 2660; if not, it RETURNs to the line after the one it just came from (2490); that is to 2500, which is another RETURN; this time to 1320.

This matter of RETURN, which always sends the computer back to the line after the one it came from, becomes difficult to follow when one GOSUB/RETURN loop is nested inside another, but with care in following it through, the programmer will find that the computer never makes a mistake.

At 1320 is GOSUB 2000. Here is the routine for moving the man along the tight-rope; just one step at a time. It is used for both the correct and wrong letters, and is printed in what I call the 'Transparent' mode: that is it does not obliterate the characters that already exist on the screen at that spot. This is achieved by using: "PRINT CHR$(22)+ CHR$(1)' at the start of the routine, and 'PRINT CHR$(22) +CHR$(0)' at its end.

In line 1120 U and V were given coordinates for the position of the man (CHR$(248)) on the tight-rope, and in 2030 a LOCATE/PRINT statement places him on the wire. 2040 is a pause to ensure he stays there long enough to be seen, and in 2050, at the same coordinates, he is erased – by using PEN 0, which is the background (or screen) colour to print him. 2060 moves his U coordinate forward by one space (U=U+1), and 2070 prints him in the new position – one step forward. 2080 is a RETURN (to 1330) to a GOTO 1130 for a new INPUT asking for a letter.

That explanation of a 'Correct letter' move has been gone into in sufficient detail to help anyone to follow the reasoning that must always be used when trying to make many things happen as the result of just one entry. If you do not yet

thoroughly understand it, then go back a page or so, and go over it again and again, until it is clear. Only in this way can you master the involved procedures needed.

If you have entered a wrong letter in response to the INPUT in line 1130, then line 1190 directs the computer to 1400. Here, a different set of location variables are used: M and Y. This is so that each succeeding wrong letter can be printed in succeeding positions. In 1120 M was given the value of 23. In 1400 Y is given the value of Y+3 (one position lower than the correct letters), and Z$ is printed there. M is then increased by 1 (so that the next letter is alongside the first, and N (which was 0 in 1120) is also increased by 1).

In 1410 comes a line that decides when too many wrong letters have been entered: IF M=30 THEN 1500 (we will explore 1500 later). It will be realised that if M is originally 23, then the addition of 7 wrong letters will make M=30.

In 1420 is a SOUND, which is different than the Sound made for a correct letter. It is followed by GOSUB 2000, which is the section to move the man farther along the wire (which we have already dealt with), followed by GOTO 1130, the INPUT asking for another letter.

The computer continues to RETURN to 1130 until either one of two conditions is met: in 2640, IF K = L (this is in the correct letter sequence); and in 1410, IF M = 30 (in the wrong letter sequence). We will deal with the correct letter sequence first. IF K = L (this has been explained earlier, so if you are still in doubt go back to it) then GOTO 2660, which says GOTO 2800, the Finish-off routine.

In 2830 L is again defined as the LENgth of A$, and a FOR/NEXT loop starts by defining U (which is the X value for moving the man) as 8 (the position on the board) +L (the LENgth of A$) +N (which in 1400 was increased by 1 every time there was a wrong letter). Thus making 8 equal to itself plus the length of the word and plus the number of wrong letters, to 22, which is the end of the tight-rope. He is left there (line 2880) and a space is printed at 23,16. In 2890, using red, 'That's right' is printed. In 2900 two lines are cleared with a PRINT SPC(17) statement, and in 2910 a SOUND is made; notice the way it is done so that a rising series of notes is delivered: a FOR/NEXT loop to define the

actual notes (in the second position after SOUND). We will ignore line 2920 for the moment, except for its final statement: GOTO 1530. At 1530 is a sequence of six lines to give a repeat game. This will be explained later.

The alternative finishing sequence, mentioned earlier, lies in line 1410 IF M=30 THEN 1500. In 1500 is a line to clear four lines in the small panel. In 1510, in red, is printed 'He fell', and in 1520, in blue, is printed a star, the correct word, and another star, followed by GOSUB 2200, which gives an impression of the man falling into the river.

Line 2230 starts a sequence of movement preceded by the transparent mode once again; it erases the man. 2240 gives O the value of 50 (for use in SOUND), U gets the value of 17, and a FOR/NEXT loop is started that gives V the value of 7 TO 16 (down the screen). 2250 prints the first man off the rope, there is a slight pause at 2260; in 2270 the man is erased; in 2280 is the first note of the SOUND: O is made equal to O+10; and NEXT V goes back to the start of the FOR/NEXT loop in line 2240, to continue printing and erasing the man until V is used up. The SOUND being made gets lower in tone as the man falls. Then, in 2290 comes a score line (to be explained later), and in 2300 a RETURN to 1530 – where is the offer of another try.

When describing the WINDOW section at 6000, it was seen that line 6050 printed 'Wrong' and 'Correct' in the small WINDOW. 6060 gave the variable WR the value of 0, and this was printed at 9,3 in the WINDOW; RI was also given the value of 0, and this was printed at 29,3. These are for the scores. In line 2290, after the man has finished falling, WR is made equal to WR+1, and that is printed at 9,3. In line 2920, after the man has successfully crossed the tight-rope, RI is made equal to RI+1, and this goes to 29,3, after CORRECT. In this way, whether the word is guessed correctly or not, one is added to the appropriate variable, and, because this is printed in the WINDOW at the screen top, it is not erased when you answer Y (yes) for another try in answer to the INPUT in line 1530. In response to Y there is a command to clear the larger WINDOW only (notice how this is worded: THEN WINDOW 1,40,3,25:CLS:GOTO 80). In response to N (no) the screen is cleared, leaving the title and score, while

'THANKS FOR THE GAME' is printed in the centre of the screen. If you should use RUN now (after ESC, of course), you will find that the score is put back to 0.

In this chapter the program 'CANYON CROSSING' has been explained at length so that the veriest beginner should be able to understand how each section works. Even more advanced programmers (who may often have used, but not understood some routines), may find them of interest.

```
10 REM ************************
20 REM **** CANYON CROSSING ****
30 REM ************************
40 INK 0,12:INK 1,25:INK 2,20,23:INK
 3,7
50 BORDER 12:PAPER 0:PEN 1:CLS
60 GOSUB 6000
70 KEY 138,"WINDOW 1,40,1,25"+CHR$(1
3)
80 GOSUB 5000
1000 REM ****************
1010 REM ***** GAME *****
1020 REM ****************
1030 RESTORE 3030
1040 B=RND*210
1050 FOR C=0 TO B
1060 READ A$
1070 NEXT C
1080 L=LEN(A$)
1090 LOCATE 23,21
1100 FOR F=1 TO L:B$=STRING$(F,45):N
EXT F
1110 PRINT B$
1120 X=23:Y=16:U=9:V=6:M=23:K=0:N=0
1130 LOCATE X,Y:INPUT "GUESS A LETTE
R";Z$
1140 Z$=UPPER$(Z$)
```

```
1150 FOR P=1 TO L
1160 IF Z$=MID$(A$,P,1) THEN 1300
1170 NEXT P
1180 FOR P=1 TO L
1190 IF Z$<>MID$(A$,P,1) THEN 1400
1200 NEXT P
1300 LOCATE X,Y+2:PRINT Z$
1310 GOSUB 2400
1320 GOSUB 2000
1330 GOTO 1130
1400 LOCATE M,Y+3:PRINT Z$;:M=M+1:N=
N+1
1410 IF M=30 THEN 1500
1420 SOUND 1,400:GOSUB 2000
1430 GOTO 1130
1500 X=23:FOR Y=16 TO 19:LOCATE X,Y:
PRINT SPC(17):NEXT Y
1510 PEN 3:LOCATE 25,17:PRINT "HE  F
ELL":PEN 1
1520 PEN 2:LOCATE 23,16:PRINT "* ";A
$;" *":PEN 1:GOSUB 2200
1530 LOCATE 23,19:INPUT "TRY AGAIN(Y
/N)";Q$
1540 IF Q$="" THEN 1540
1550 IF Q$="Y" OR Q$="y" THEN WINDOW
 1,40,4,25:CLS:GOTO 80
1560 IF Q$="N" OR Q$="n" THEN CLS
1570 LOCATE 10,12:PRINT "THANKS FOR
THE GAME"
1580 GOTO 1580
2000 REM ************************
2010 REM *** MOVING THE MAN ***
2020 REM ************************
2030 PRINT CHR$(22)+CHR$(1):LOCATE U
,V:PRINT CHR$(248)
2040 FOR W=1 TO 500:NEXT W
```

```
2050 PEN 0:LOCATE U,V:PRINT CHR$(248
):PEN 1
2060 U=U+1
2070 LOCATE U,V:PRINT CHR$(248):PRIN
T CHR$(22)+CHR$(0)
2080 RETURN
2200 REM ************************
2210 REM ***** MAN FALLING *****
2220 REM ************************
2230 PRINT CHR$(22)+CHR$(1):PEN 0:FO
R U=11 TO 21:LOCATE U,V:PRINT CHR$(2
48):NEXT U:PEN 1
2240 O=50:U=17:FOR V=7 TO 16
2250 LOCATE U,V:PRINT CHR$(248)
2260 FOR W=1 TO 50:NEXT W
2270 PEN 0:LOCATE U,V:PRINT CHR$(248
):PEN 1:PRINT CHR$(22)+CHR$(0)
2280 SOUND 1,0,10:O=O+10:NEXT V
2290 WR=WR+1:LOCATE #1,9,3:PRINT #1,
WR
2300 RETURN
2400 REM ***************************
2410 REM ***** PLACE LETTERS *****
2420 REM ***************************
2430 X=23
2440 IF INSTR(1,A$,Z$)>0 THEN 2450
2450 FOR P=1 TO L
2460 LOCATE P+X-1,Y+5
2470 IF MID$(A$,P,1)=Z$ THEN PRINT Z
$;
2480 SOUND 1,200,5:NEXT P
2490 K=K+1:GOSUB 2600
2500 RETURN
2600 REM ***************************
2610 REM ***** COMPLETE WORD *****
2620 REM ***************************
```

```
2630 L=LEN(A$)
2640 IF K=L THEN 2660
2650 RETURN
2660 GOTO 2800
2800 REM ************************
2810 REM ***** FINISH OFF *****
2820 REM ************************
2830 L=LEN(A$):FOR U=8+L+N TO 22
2840 PRINT CHR$(22)+CHR$(1):LOCATE U
,V:PRINT CHR$(248)
2850 FOR W=1 TO 500:NEXT W
2860 PEN 0:LOCATE U,V:PRINT CHR$(248
):PEN 1:PRINT CHR$(22)+CHR$(0)
2870 NEXT U
2880 LOCATE U,V:PRINT CHR$(248):LOCA
TE 23,16:PRINT SPC(17)
2890 PEN 3:LOCATE 23,17:PRINT "THAT'
S RIGHT      ":PEN 1
2900 FOR Y=18 TO 19:LOCATE 23,Y:PRIN
T SPC(17):NEXT Y
2910 FOR G=100 TO 50 STEP -5:SOUND 1
,G,10:NEXT G
2920 RI=RI+1:LOCATE #1,29,3:PRINT #1
,RI:GOTO 1530
3000 REM ************************
3010 REM ***** DATA STORE *****
3020 REM ************************
3030 DATA ABDUCT,ABOUT,ABHOR,ABIDE,A
BDOMEN,ABJECT,ABJURE,ABLUTION,ABSOLU
TE,ABOLISH
3040 DATA ACERBITY,ACETIFY,ACOLYTE,A
COUIRE,ACRIMONY,ACRONYM,ACTING,ACTIV
E,ACTOR,ACUITY
3050 DATA ADHESION,ADIPOSE,ADJOURN,A
DJUNCT,ADMIRE,ADMONISH,ADOPTIVE,ADOR
N,ADULTERY,ADVERSITY
```

```
3060 DATA BABYLON,BACHELOR,BACKING,B
ACTERIUM,BADGER,BAILEY,BALCONY,BALUS
TER,BANGLE,BANISTER
3070 DATA BANKRUPTCY,BAPTISM,BARELY,
BARITONE,BARONETCY,BAROUCHE,BASHFUL,
BASINET,BASKET,BASTION
3080 DATA BEAUFORT,BEACON,BEADING,BE
ARD,BEAUTIFY,BECKON,BEDOUIN,BEGONIA,
BIOGRAPHY,BIOPLASM
3090 DATA CABINET,CABRIOLET,CADGER,C
AIRN,CALDRON,CALORIE,CALUMET,CALYPSO
,CAMEO,CAMISOLE
3100 DATA CEANOTHUS,CEDAR,CENOTAPH,C
ENTAUR,CENTRALISM,CHAGRIN,CHALET,CHA
MPION,CHANDLER,CHAPEL
3110 DATA CHEMISTRY,CHIEFLY,CHIMERA,
CHINTZ,CHISEL,CHIVALROUS,CHLORINATE,
CHOLERA,CHUTNEY,CINEMA
3120 DATA DAINTY,DAISY,DARLING,DAUGH
TER,DAYLIGHT,DEACON,DEBAUCH,DEBONAIR
,DECAPOD,DECIMAL
3130 DATA DEFAULT,DEFIANT,DEFORMITY,
DELIGHT,DELUSION,DEMOCRAT,DENIAL,DEN
SITY,DEPART,DEPOSIT
3140 DATA DIAGNOSE,DIALECT,DIALOGUE,
DISCOVERY,DISCOUNT,DOCILE,DOCTRINAL,
DOLPHIN,DOMINATE,DRAUGHT
3150 DATA EARTHLY,EARWIG,EBONY,ECHID
NA,EDICT,EDITOR,EDUCATION,EGOTISM,EG
YPTIAN,EIGHTY
3160 DATA ELASTIC,ELBOW,ELDRITCH,ELO
CUTARY,ELYSIUM,EMBARGO,EMBANK,EMBLAZ
ON,EMBODY,EMBRYO
```

```
3170 DATA EMBRYONIC,EMPATHY,EMPHATIC
,EMPLOY,EMPORIUM,EMPTY,EMULSIFY,ENCR
UST,ENDOW,ENFOLD
3180 DATA FABLE,FABRIC,FACETIOUS,FAC
TORY,FEAST,FEIGN,FELONY,FENCING,FERA
L,FESTIVAL
3190 DATA FETLOCK,FEUDAL,FIBULA,FICK
LE,FIDGET,FIELD,FIERY,FIGMENT,FIGURA
TIVE,FILAMENT
3200 DATA FLAMING,FLANGE,FLASHING,FL
EMISH,FLIGHT,FLOATING,FRAGMENT,FREAK
,FRESCO,FRIGHTEN
3210 DATA GAELIC,GAINFUL,GALOSH,GAMB
IT,GAMBOGE,GANDER,GANTRY,GAOLER,GARD
EN,GARLIC
3220 DATA GARMENT,GEISHA,GELATINOUS,
GELDING,GENIUS,GENOA,GENTIAN,GERMAN,
GHASTLY,GHERKIN
3230 DATA GLABROUS,GLACIER,GLAMOUR,G
LAZIER,GLEAN,GLIMPSE,GLOAMING,GLORIF
Y,GLOWER,GNARLED
5000 REM ************************
5010 REM ***** DRAW SCENE *****
5020 REM ************************
5030 WINDOW 1,40,4,25
5040 ORIGIN 0,48:DRAW 128,128,3:DRAW
R 16,32:DRAWR 0,96:DRAWR -16,16:DRAW
R -32,16:DRAWR -96,0
5050 ORIGIN 144,304:DRAW -144,0:ORIG
IN 144,208:DRAW -144,0:ORIGIN 128,17
6:DRAW -128,0:ORIGIN 96,144:DRAW -96
,0:ORIGIN 64,112:DRAW -64,0:ORIGIN 3
2,80:DRAW -32,0
```

```
5060 ORIGIN 32,80:DRAW 16,-82:ORIGIN
 64,112:DRAW 16,-104:ORIGIN 96,144:D
RAW 16,-76:ORIGIN 128,176:DRAW 16,-6
8:ORIGIN 144,208:DRAW 16,-80:ORIGIN
144,304:DRAW 16,-96
5070 ORIGIN 176,0:DRAW 144,144:ORIGI
N 320,144:DRAW 16,32:ORIGIN 336,176:
DRAW 0,96:ORIGIN 336,272:DRAW -64,64
:ORIGIN 336,272:DRAW -16,-80:ORIGIN
304,304:DRAW -16,-64:ORIGIN 240,304:
DRAW 8,32
5080 ORIGIN 272,336:DRAW -14,-52:ORI
GIN 272,336:DRAW 96,0:ORIGIN 304,304
:DRAW 64,0:ORIGIN 336,272:DRAW 32,0:
ORIGIN 336,176:DRAW 32,0:ORIGIN 304,
128:DRAW 48,0:ORIGIN 272,96:DRAW 48,
0:ORIGIN 208,32:DRAW 112,0:ORIGIN 24
0,64:DRAW 80,0
5090 ORIGIN 639,8:DRAW -296,0,1:DRAW
R 0,112:DRAWR 296,0:DRAWR 0,-112
5100 ORIGIN 80,0:DRAW 32,64,2:ORIGIN
 112,64:DRAWR 48,64:ORIGIN 162,128:D
RAW 2,80:ORIGIN 320,148:DRAW 0,44:OR
IGIN 320,194:DRAW -16,32:ORIGIN 304,
224:DRAW -96,112
5110 ORIGIN 104,8:DRAW 32,64,2:ORIGI
N 156,84:DRAW 24,40:ORIGIN 188,152:D
RAW 0,48:ORIGIN 176,224:DRAW -12,18
5120 ORIGIN 144,32:DRAW 32,48:ORIGIN
 192,112:DRAW 16,48:ORIGIN 208,192:D
RAW -16,48:ORIGIN 176,256:DRAW -16,1
6
```

```
5130 ORIGIN 160,16:DRAW 24,48:ORIGIN
 200,64:DRAW 24,48:ORIGIN 232,144:DR
AW 0,32:ORIGIN 224,203:DRAW -16,32:O
RIGIN 192,256:DRAW -16,16:ORIGIN 160
,288:DRAW -16,16
5140 ORIGIN 190,16:DRAW 16,32:ORIGIN
 222,48:DRAW 16,48:ORIGIN 248,112:DR
AW 8,32:ORIGIN 256,176:DRAW -10,32:O
RIGIN 240,224:DRAW -16,24:ORIGIN 208
,272:DRAW -16,16:ORIGIN 176,304:DRAW
 -16,16
5150 ORIGIN 272,98:DRAW 8,32:ORIGIN
274,144:DRAW 0,32:ORIGIN 272,192:DRA
W -16,32:ORIGIN 256,240:DRAW -32,32:
ORIGIN 208,288:DRAW -16,16
5160 ORIGIN 292,122:DRAW 4,34:ORIGIN
 312,160:DRAW 0,32:ORIGIN 300,176:DR
AW -12,32:ORIGIN 288,224:DRAW -16,16
:ORIGIN 256,264:DRAW -16,16:ORIGIN 2
24,288:DRAW -16,16:ORIGIN 192,320:DR
AW -16,16
5170 ORIGIN 0,238:DRAW 128,0,1:DRAWR
 0,16:DRAWR 224,0:DRAWR 0,-16:DRAWR
32,0:ORIGIN 128,256:DRAW -32,-16:ORI
GIN 352,256:DRAW 32,-16
5180 LOCATE 25,3:PRINT "Guess a word
 to":LOCATE 25,4:PRINT "help a man o
ver":LOCATE 25,5:PRINT "the tight-ro
pe."
5190 LOCATE 26,6:PRINT "A wrong"
```

```
5200 LOCATE 26,7:PRINT "letter causes":LOCATE 26,8:PRINT "trouble.":LOCATE 25,10:PRINT "SEVEN LETTERS":LOCATE 27,12:PRINT "WRONG ARE"
5210 LOCATE 29,14:PRINT "FATAL"
5220 LOCATE 31,18:PRINT CHR$(242);" CORRECT":LOCATE 33,19:PRINT CHR$(242);" WRONG"
5230 LOCATE 1,1:PRINT STRING$(40,238)
5240 ORIGIN 0,336:DRAW 0,-336:DRAWR 639,0:DRAWR 0,336
5250 RETURN
6000 REM ********************
6010 REM ***** WINDOW *****
6020 REM ********************
6030 WINDOW #1,1,40,1,3:PEN #1,2
6040 LOCATE #1,2,1:PRINT #1,"* *  C A N Y O N  C R O S S I N G  * *"
6050 LOCATE #1,2,3:PRINT #1,"WRONG:          CORRECT:"
6060 WR=0:LOCATE #1,9,3:PRINT #1,WR
6070 RI=0:LOCATE #1,29,3:PRINT #1,RI
6080 RETURN
```

* * CANYON CROSSING * *
WRONG! 0
RIGHT! 0
Guess a word to help a man over the tight-rope. A wrong letter causes trouble.
SEVEN LETTERS
WRONG ARE FATAL
GUESS A LETTER
← CORRECT
↓ WRONG
_ _ _ _ _ _ _

Chapter Ten

PRINTING A SCREEN DISPLAY ON THE DMP1

This final chapter is devoted to explaining a BASIC Screen Dump program; one that will produce a black and white copy of any screen display on the printer. Because it is written entirely in BASIC, and entails a great deal of storing into memory, it is quite slow in action (it takes about ten minutes to produce a full and detailed screen display). On the other hand it is simple, and very easy to manage.

The program (at this chapter end) works perfectly on the DMP1, but might need some alterations to work on other printers. It has been placed at lines 50000 to 50270, so that it will be well above most programs, and therefore can be MERGEd easily. It is quite short, and easy to enter, but it must be typed in exactly as shown in the print-out listing. This, like all the listings in this book, is printed on the DMP1 from a working program, so if it is copied exactly it is bound to work. If errors crop up when it is RUN, then they must be typing errors, or faults in the computer or printer. We will describe the program first, and how it is used to obtain the screen dump later.

Line 50030 is a REMinder of the entry to be made in the program to be copied (explained later). 50040 defines the array to be set up in the computer to contain one printer line of a 320 bit pattern; it is given the variable A%. At this point it should be understood that the computer, when printing any character on the screen uses 8 rows of bits (from top to bottom of the character). You have met this characteristic when making "User Defined Characters" with the aid of SYMBOL. The printer, because of the way it uses its pins, uses 7 rows of bits. To obtain a representation of whatever is on the screen, the computer scans the screen in 7 horizontal rows; places that in memory, and then proceeds to print (from left to right) all the bits it has in memory. It is this memorising that takes all the time; the printing itself is fairly fast.

Line 50050 is needed to prepare the computer for dot graphic printing. If it was not decreed as 255, a default width of 132 would be assumed, and that would not be sufficient. 50060 sets the variable T at 413. In 50070 a FOR/NEXT loop starts using B%=0 TO 28 (28 is the number of rows of 7 bits each, from top to bottom of the screen). In 50080 T is reduced by 14, and in 50090 another FOR/NEXT loop is started for C=0 TO 319 (the number of dots from left to right in the row).

Then comes 50100 GOSUB 50200, and there we have the search routines that use TEST. Lines 50200 to 50260 define the S1% to S7% as the contents (in dots) of each of the 7 rows. It will be seen that T reduces by 2 in each row, and that each succeeding row is multiplied by 2, 4, 8, 16, 32, and 64. This goes into memory, and 50190 RETURN sends the computer back to 50110 where A%(C%) is now given the value of all those S1% to S7% lines. The NEXT in line 50120 loops back to 50090 (FOR C%) so that all the S% lines are included.

In lines 50130 to 50150 come the PRINT#8 commands to the printer. CHR$(27) and CHR$(75) is a graphic printing specification; CHR$(2) is a line feed and carriage return; and CHR$(64) prepares the printer for 320 bits to print. Lines 50140–50150 do the actual printing with a FOR/NEXT loop using D% 0 TO 320. 50160 NEXT loops back to 50070 to start the sequence for the next line of printing, until all the rows are complete.

Line 50170 is a command to finish all dumping, and to reset the printer to its original mode (for listing or text); 50180 erases any residue of graphic printing commands from memory, and 50190 is a RETURN to the main program: to a line that has yet to be explained: 50030 REM.

If you have now entered this program correctly, and have saved it on tape, you will want to try it out. We will use the program CROSS-JOIN WORDS (Chapter Seven) as an example. Use MERGE "" to enter it; do not use LOAD "". Now RUN that program, and press G to get the game display on the screen. This is what we shall print out on the printer. To do this LIST the main program until you reach line 1040. This is the INPUT asking for a word, and it stops the computer until a word is entered. It is the position we want.

Because INPUT holds the computer for a word, we must first of all alter INPUT to PRINT, and then delete the semicolon and variable (;W$). Then enter a new line (the one that is in the REM line at 50030). So the line is: ORIGIN 0,0: GOSUB 50000:END. The ORIGIN command is essential to put the computer into graphic mode, and the 0,0 is needed to locate the graphic cursor at its resting position.

Then comes GOSUB 50000, which sends the computer to the new DUMP program, and, after ten minutes of waiting and printing (28 times) the word Ready will flash up on the screen; the computer will have gone back to the main program, and will have found the word 'END'. Don't forget that if you want to play the game of CROSS-JOIN now you will need to delete line 1045, and replace the word INPUT (instead of PRINT) in 1040, as well as replacing the semicolon and variable at its end. There is a print-out of the game screen (just after the program at this chapter end) that was obtained by this program.

Remember these important points: always use ORIGIN 0,0: before the GOSUB 50000:, and finish that line with END. This can be placed anywhere you want in any program of which you want to make a print. For instance, in that CROSS-JOIN program it could be at line 4115 to obtain a copy of the first screenful of instructions. In that case there would be no need to alter the previous line, because it is already set to PRINT in line 4110.

Another point: when the item you wish to obtain is on the screen, and you are waiting for the printer to spring into action, don't be impatient – it takes a long time to start. It takes at least 20 seconds for the first line of print, and this is a very long time when one is waiting for the first trial. Therefore be patient: unless you get an error message on the screen, it will eventually start. If an error message appears then LIST 50000 – and check the entries. Presumably you already know that the dash after the number makes it continue to scroll the list.

Never stop the printer in the middle of a DUMP procedure. If you do it is essential to use a direct entry of 'PRINT#8, CHR$(15)' before you try listing anything on the computer. Also directly enter PRINT#8,CHR$(27)"@". The @ sign

between those inverted commas (after (27)) is on the key just after P. That entry will clear the printer of any hang-over instructions.

If this chapter has helped you handle the DMP1 a little better, its purpose will have been achieved. Read the Printer Manual thoroughly; pages 9 to 19 are full of information that makes more sense as you progress. On page 12, for instance, those numbers down the left-hand side are CHR$ numbers (some of which are met in the DUMP program). Keep on studying.

```
50000 REM ************************
50010 REM ***** BASIC DUMP *****
50020 REM ************************
50030 REM ORIGIN 0,0:GOSUB 50000:END
50040 DIM A%(320)
50050 WIDTH 255
50060 T=413
50070 FOR B%=0 TO 28
50080 T=T-14
50090 FOR C%=0 TO 319
50100 GOSUB 50200
50110 A%(C%)=S1%+S2%+S3%+S4%+S5%+S6%
+S7%
50120 NEXT
50130 PRINT#8,CHR$(27);CHR$(75);CHR$
(2);CHR$(64);
50140 FOR D%=0 TO 320:PRINT#8,CHR$(A
%(D%));:NEXT
50150 PRINT#8,CHR$(A%(320))
50160 NEXT
50170 PRINT#8,CHR$(15)
50180 ERASE A%
50190 RETURN
50200 S1%=-1*(TEST(C%*2,T)>0)
50210 S2%=-1*(TEST(C%*2,T-2)>0)*2
```

```
50220 S3%=-1*(TEST(C%*2,T-4)>0)*4
50230 S4%=-1*(TEST(C%*2,T-6)>0)*8
50240 S5%=-1*(TEST(C%*2,T-8)>0)*16
50250 S6%=-1*(TEST(C%*2,T-10)>0)*32
50260 S7%=-1*(TEST(C%*2,T-12)>0)*64
50270 RETURN
```

CROSSJOIN WORDS
ABCDEFGHIJKLMNO
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
*
L 17 LETTERS
THIS ARE GIVEN
K
E
Use any of these
letters to make
a word, & then
ENTER IT COMPLETE.
Then locate place
for 1st letter,
& then:-
PRESS Top letter.
PRESS side letter.
PRESS A(across)
or D(down)
K J U A P L E I B T D S H G W O Y
ENTER WORD

INDEX

Page

ALPHABETICAL SORT 21
ALTERATIONS 67
ANAGRAMMATISM 6
ANAGRAMS 1, 40
CANYON CROSSING 75, 82
CROSS-JOIN WORDS 55, 59
CROSSED LINES ROUTINE 12
CROSSWORD SOLVER 48, 51
DMP1 92
ERASING 57
GOSUB/RETURN 2, 34, 79
GUESS THE WORD 40, 44
HASH 21, 75
HOW MANY WORDS 32, 36
IF SEQUENCE 43
INPUT/PRINT 57
KEY DEFINING 76
LEFT$ 3, 31
LEN 3, 66
LETTER EATER 29, 34
LETTER EXTRACTION 4, 13
MID$ 3
MOVE A LETTER 15
MOVE BALL 31
MULTIPLE ANSWERS 56
MULTIPLE OUTPUTS 13
PLOTTING BOARD 10
PRINTER 24, 32, 58
RANDOM DATA 44
RANDOM SORT/COLOURS 29, 30
RANDOMIZE TIME 58
READ/DATA 23, 41, 42
RENUM 22
RIGHT$ 3, 31
RND 3
SCORING 33
SCREEN DUMP PROCEDURE 93, 94

Page
SCREEN DUMP 92, 95
SHUFFLE FORMULA 8
SOUND 80
SPACING LETTERS 68
SPC 5
SQUARE OF LETTERS 12
STRING$ 48
TAG/TAGOFF 58, 68
THINK OF A WORD SQUARE 64, 69
TRANSPARENT MODE 79
UPPER/LOWER 44, 55
VARIABLES 66
VARIOUS ROUTINES 50
WHILE/WEND 20
WINDOW USE 20, 40, 75
WORD SORT 25

Please note following is a list of other titles that are available in our range of Radio, Electronics and Computer Books.

These should be available from all good Booksellers, Radio Component Dealers and Mail Order Companies.

However, should you experience difficulty in obtaining any title in your area, then please write directly to the Publisher enclosing payment to cover the cost of the book plus adequate postage.

If you would like a complete catalogue of our entire range of Radio, Electronics and Computer Books then please send a Stamped Addressed Envelope to:–

BERNARD BABANI (publishing) LTD
THE GRAMPIANS
SHEPHERDS BUSH ROAD
LONDON W6 7NF
ENGLAND

160	Coil Design and Construction Manual	£1.95
202	Handbook of Integrated Circuits (IC's) · Equivalents & Substitutes	£1.95
205	First Book of Hi-Fi Loudspeaker Enclosures	£0.95
208	Practical Stereo and Quadrophony Handbook	£0.75
214	Audio Enthusiasts Handbook	£0.85
219	Solid State Novelty Projects	£0.85
220	Build Your Own Solid State Hi-Fi and Audio Accessories	£0.85
221	28 Tested Transistor Projects	£1.25
222	Solid State Short Wave Receivers for Beginners	£1.95
223	50 Projects Using IC CA3130	£1.25
224	50 CMOS IC Projects	£1.35
225	A Practical Introduction to Digital IC's	£1.75
226	How to Build Advanced Short Wave Receivers	£1.95
227	Beginners Guide to Building Electronic Projects	£1.95
228	Essential Theory for the Electronics Hobbyist	£1.95
RCC	Resistor Colour Code Disc	£0.20
BP1	First Book of Transistor Equivalents and Substitutes	£1.50
BP2	Handbook of Radio, TV, Ind & Transmitting Tube & Valve Equivalents	£0.60
BP6	Engineers and Machinists Reference Tables	£0.75
BP7	Radio and Electronic Colour Codes and Data Chart	£0.40
BP14	Second Book of Transistor Equivalents & Substitutes	£1.75
BP24	52 Projects Using IC741	£1.75
BP27	Chart of Radio, Electronic, Semi-conductor and Logic Symbols	£0.50
BP28	Resistor Selection Handbook	£0.60
BP29	Major Solid State Audio Hi-Fi Construction Projects	£0.85
BP32	How to Build Your Own Metal and Treasure Locators	£1.95
BP33	Electronic Calculator Users Handbook	£1.50
BP34	Practical Repair and Renovation of Colour TVs	£1.95
BP36	50 Circuits Using Germanium, Silicon and Zener Diodes	£1.50
BP37	50 Projects Using Relays, SCR's and TRIACS	£1.95
BP39	50 (FET) Field Effect Transistor Projects	£1.75
BP42	50 Simple L.E.D. Circuits	£1.50
BP43	How to Make Walkie-Talkies	£1.95
BP44	IC 555 Projects	£1.95
BP45	Projects in Opto-Electronics	£1.95
BP47	Mobile Discotheque Handbook	£1.35
BP48	Electronic Projects for Beginners	£1.95
BP49	Popular Electronic Projects	£1.95
BP50	IC LM3900 Projects	£1.35
BP51	Electronic Music and Creative Tape Recording	£1.95
BP52	Long Distance Television Reception (TV-DX) for the Enthusiast	£1.95
BP53	Practical Electronics Calculations and Formulae	£2.95
BP54	Your Electronic Calculator and Your Money	£1.35
BP55	Radio Stations Guide	£1.75
BP56	Electronic Security Devices	£1.95
BP57	How to Build Your Own Solid State Oscilloscope	£1.95
BP58	50 Circuits Using 7400 Series IC's	£1.75
BP59	Second Book of CMOS IC Projects	£1.95
BP60	Practical Construction of Pre-amps, Tone Controls, Filters & Attenuators	£1.95
BP61	Beginners Guide To Digital Techniques	£0.95
BP62	Elements of Electronics – Book 1	£2.95
BP63	Elements of Electronics – Book 2	£2.25
BP64	Elements of Electronics – Book 3	£2.25
BP65	Single IC Projects	£1.50
BP66	Beginners Guide to Microprocessors and Computing	£1.95
BP67	Counter, Driver and Numeral Display Projects	£1.75
BP68	Choosing and Using Your Hi-Fi	£1.65
BP69	Electronic Games	£1.75
BP70	Transistor Radio Fault-Finding Chart	£0.50
BP71	Electronic Household Projects	£1.75
BP72	A Microprocessor Primer	£1.75
BP73	Remote Control Projects	£1.95
BP74	Electronic Music Projects	£1.75
BP75	Electronic Test Equipment Construction	£1.75
BP76	Power Supply Projects	£1.95
BP77	Elements of Electronics – Book 4	£2.95

BP78	Practical Computer Experiments	£1.75
BP79	Radio Control for Beginners	£1.75
BP80	Popular Electronic Circuits – Book 1	£1.95
BP81	Electronic Synthesiser Projects	£1.75
BP82	Electronic Projects Using Solar Cells	£1.95
BP83	VMOS Projects	£1.95
BP84	Digital IC Projects	£1.95
BP85	International Transistor Equivalents Guide	£2.95
BP86	An Introduction to BASIC Programming Techniques	£1.95
BP87	Simple L.E.D. Circuits – Book 2	£1.35
BP88	How to Use Op-Amps	£2.25
BP89	Elements of Electronics – Book 5	£2.95
BP90	Audio Projects	£1.95
BP91	An Introduction to Radio DXing	£1.95
BP92	Easy Electronics – Crystal Set Construction	£1.75
BP93	Electronic Timer Projects	£1.95
BP94	Electronic Projects for Cars and Boats	£1.95
BP95	Model Railway Projects	£1.95
BP96	C B Projects	£1.95
BP97	IC Projects for Beginners	£1.95
BP98	Popular Electronic Circuits – Book 2	£2.25
BP99	Mini-Matrix Board Projects	£1.95
BP100	An Introduction to Video	£1.95
BP101	How to Identify Unmarked IC's	£0.65
BP102	The 6809 Companion	£1.95
BP103	Multi-Circuit Board Projects	£1.95
BP104	Electronic Science Projects	£2.25
BP105	Aerial Projects	£1.95
BP106	Modern Op-Amp Projects	£1.95
BP107	30 Solderless Breadboard Projects – Book 1	£2.25
BP108	International Diode Equivalents Guide	£2.25
BP109	The Art of Programming the 1K ZX81	£1.95
BP110	How to Get Your Electronic Projects Working	£1.95
BP111	Elements of Electronics – Book 6	£3.50
BP112	A Z-80 Workshop Manual	£2.75
BP113	30 Solderless Breadboard Projects – Book 2	£2.25
BP114	The Art of Programming the 16K ZX81	£2.50
BP115	The Pre-Computer Book	£1.95
BP116	Electronic Toys Games & Puzzles	£2.25
BP117	Practical Electronic Building Blocks – Book 1	£1.95
BP118	Practical Electronic Building Blocks – Book 2	£1.95
BP119	The Art of Programming the ZX Spectrum	£2.50
BP120	Audio Amplifier Fault-Finding Chart	£0.65
BP121	How to Design and Make your Own P.C.B.s	£1.95
BP122	Audio Amplifier Construction	£2.25
BP123	A Practical Introduction to Microprocessors	£2.25
BP124	Easy Add-on Projects for Spectrum ZX81 & Ace	£2.75
BP125	25 Simple Amateur Band Aerials	£1.95
BP126	BASIC & PASCAL in Parallel	£1.50
BP127	How to Design Electronic Projects	£2.25
BP128	20 Programs for the ZX Spectrum & 16K ZX81	£1.95
BP129	An Introduction to Programming the ORIC-1	£1.95
BP130	Micro Interfacing Circuits – Book 1	£2.25
BP131	Micro Interfacing Circuits – Book 2	£2.25
BP132	25 Simple Shortwave Broadcast Band Aerials	£1.95
BP133	An Introduction to Programming the Dragon 32	£1.95
BP134	Easy Add-on Projects for Commodore 64 & Vic-20	£2.50
BP135	The Secrets of the Commodore 64	£2.50
BP136	25 Simple Indoor and Window Aerials	£1.95
BP137	BASIC & FORTRAN in Parallel	£1.95
BP138	BASIC & FORTH in Parallel	£1.95
BP139	An Introduction to Programming the BBC Model B Micro	£2.50
BP140	Digital IC Equivalents and Pin Connections	£3.95
BP141	Linear IC Equivalents and Pin Connections	£3.95
BP142	An Introduction to Programming the Acorn Electron	£1.95
BP143	An Introduction to Programming the Atari 600XL and 800XL	£2.50
BP144	Further Practical Electronics Calculations and Formulae	£3.75
BP145	25 Simple Tropical and M.W. Band Aerials	£1.95

BERNARD BABANI BP175

How to Write Word Game Programs for the Amstrad CPC 464, 664 and 6128

- This book includes ways to sort words alphabetically, to scramble them into anagrams, to select letters from words for various purposes, to locate missing letters from words, to place letters randomly on the screen and even ways to help in writing or solving crossword puzzles.

- The many routines used for all these different ways of handling words or letters are explained clearly, and are inserted into the programs in such a way that they can easily be extracted for use in games of your own devising.

- The penultimate chapter describes a program called "Canyon Crossing" which shows how these various routines can be combined to make a fairly complex game. The final chapter is devoted to a screen dump program which enables you to make a copy of the screen on your DMP1 printer.

- Recommended for all Amstrad users be they beginners or seasoned programmers.

GB £ NET +002.95

ISBN 0-85934-149-6

00295

£2.95

9 780859 341493

Document numérisé avec amour par

AMSTRAD CPC

MÉMOIRE ÉCRITE

https://acpc.me/